The design of the molsturm package

Lessons learned from Finite Element based quantum chemistry

Michael F. Herbst
michael.herbst@iwr.uni-heidelberg.de
http://blog.mfhs.eu

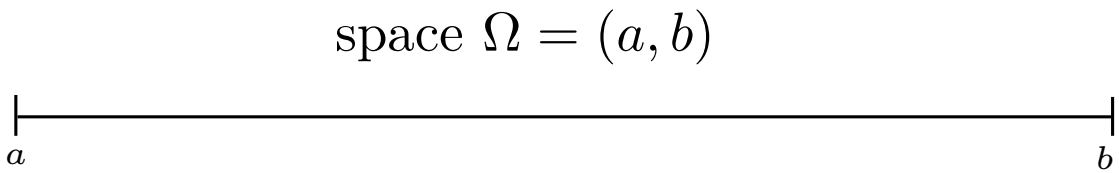
Interdisziplinäres Zentrum für wissenschaftliches Rechnen
Ruprecht-Karls-Universität Heidelberg

9th December 2016

Contents

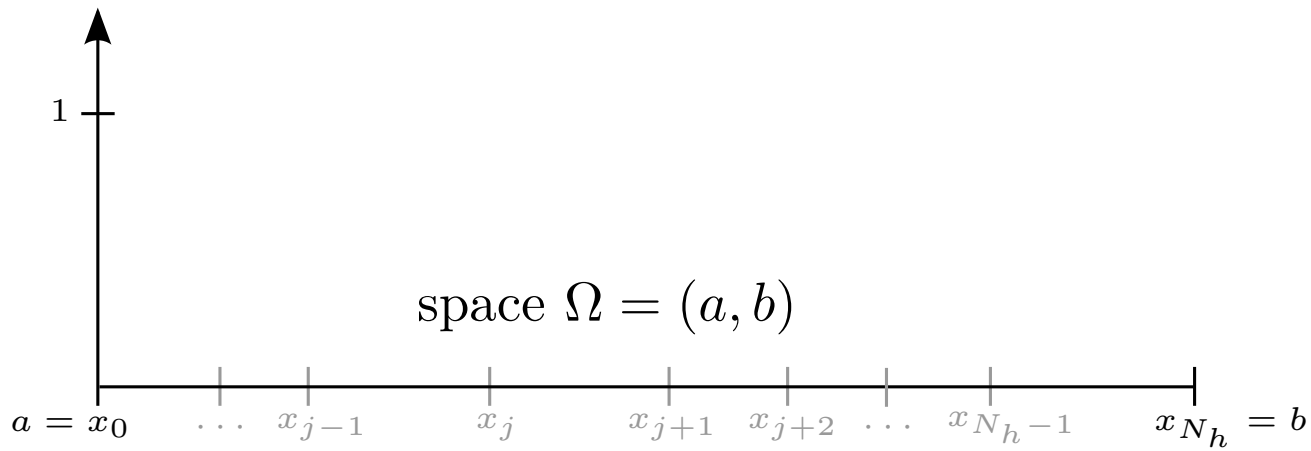
Contents

Finite elements (FEs) as basis functions



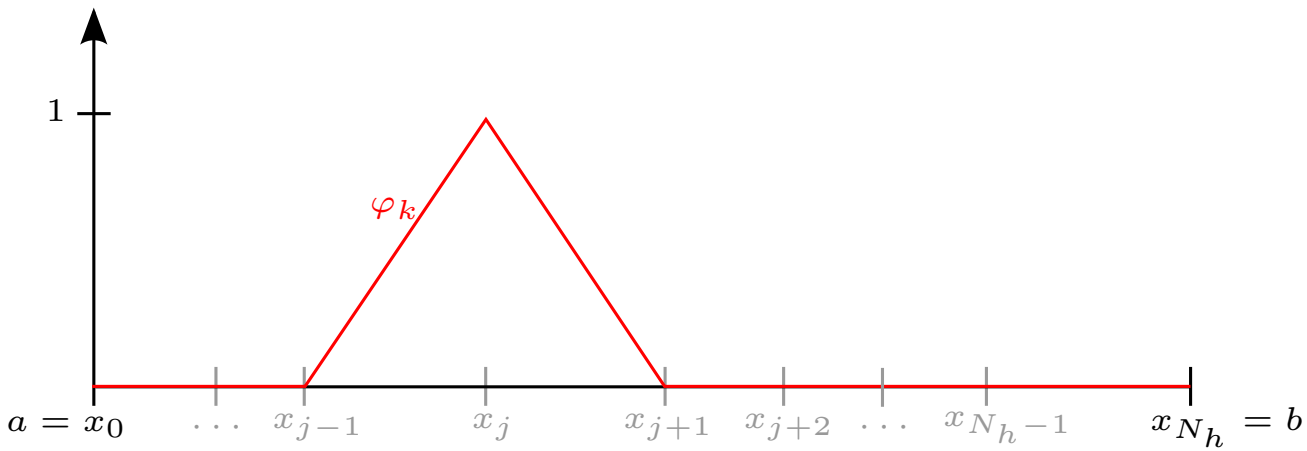
- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

Finite elements (FEs) as basis functions



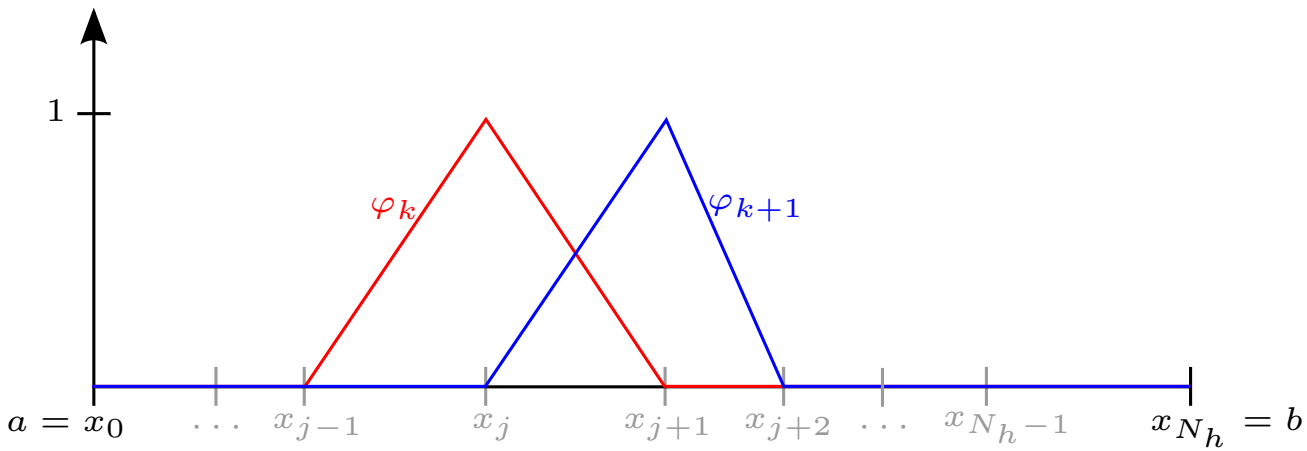
- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

Finite elements (FEs) as basis functions



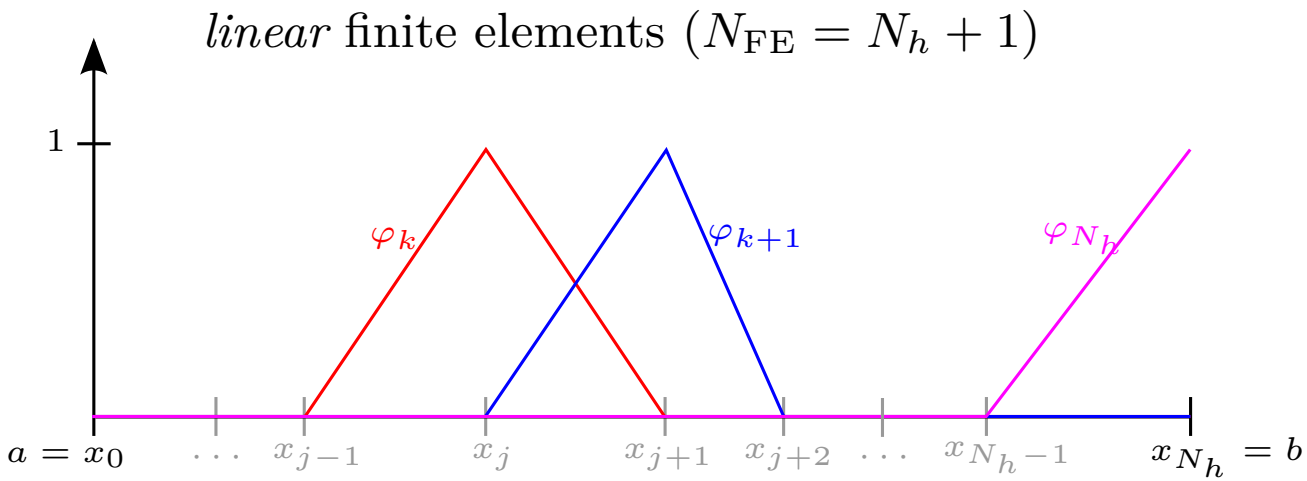
- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

Finite elements (FEs) as basis functions



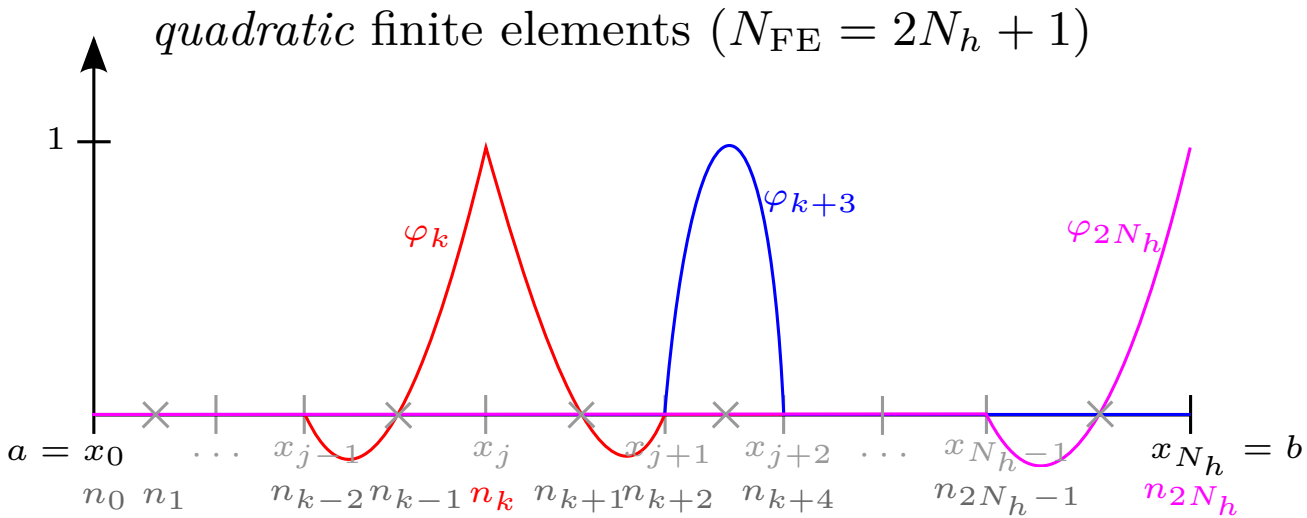
- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

Finite elements (FEs) as basis functions



- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

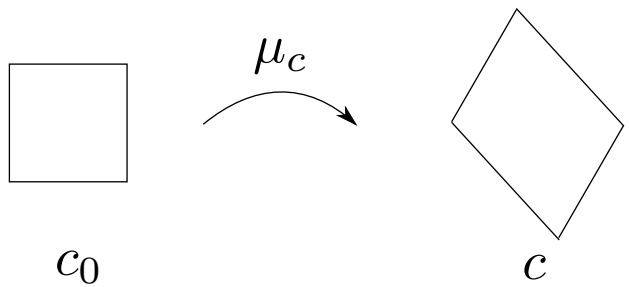
Finite elements (FEs) as basis functions



- Discretise open set Ω into grid of N_h cells.
- Support on just a few neighbouring cells
- At *nodal points*: $\varphi_i(n_j) = \delta_{ij}$
- N_{FE}-dim. basis for discretised Hilbert space H_0^1.

Reference cell and shape functions

- Cells c can have arbitrary shape
- Related to *reference cell* $c_0 = [0,1]^3$ by a map μ_c

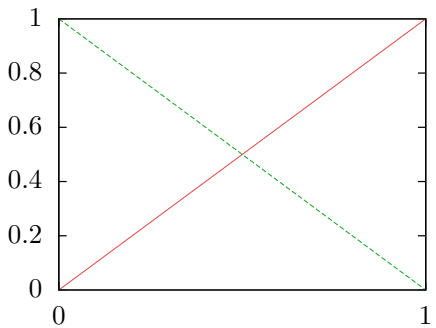


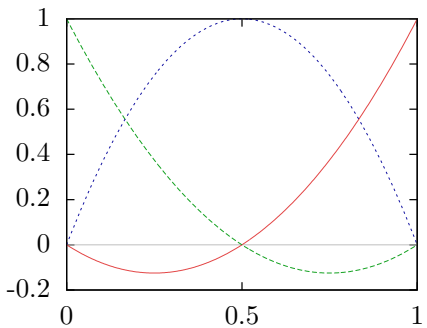
- All FEs φ_k constructed from *shape functions* $\{e_\alpha\}$ and $\{\mu_c\}$

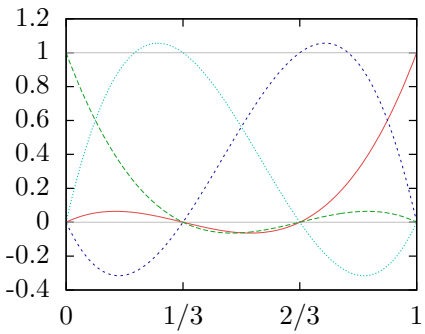
$\Rightarrow$ Computation on the FE grid:

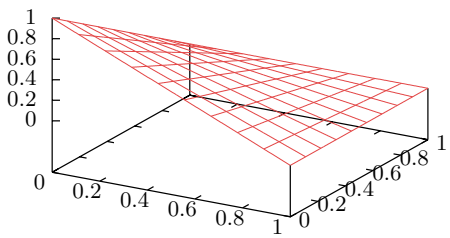
- Compute on reference cell once
- Transform result onto all grid cells

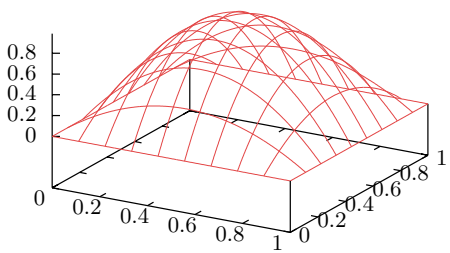
Examples of shape functions

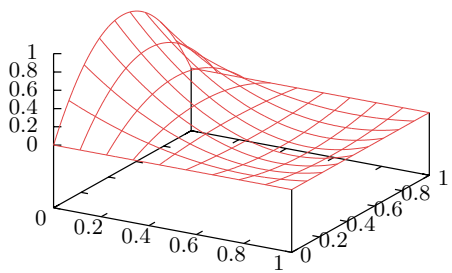


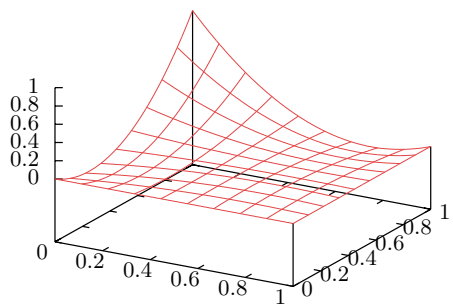












Notes and observations

- Locality built into the basis
- Need large number of basis functions (millions)

$\Rightarrow$ Need linear scaling in N_{FE}

- Arbitrary boundary conditions
- Arbitrary mesh shapes
- Error estimation and adaptive refinement

$\Rightarrow$ No bias towards expected solution

Notes and observations

- Locality built into the basis
- Need large number of basis functions (millions)

$\Rightarrow$ Need linear scaling in N_{FE}

- Arbitrary boundary conditions
- Arbitrary mesh shapes
- Error estimation and adaptive refinement

$\Rightarrow$ No bias towards expected solution

Formulation of the problem

- The well-known canonical Hartree-Fock equations ...

$$\Big(-\frac{1}{2}\Delta + \hat{\mathcal{V}}(\underline{\boldsymbol{r}})\Big)\psi_i(\underline{\boldsymbol{r}}) = \varepsilon_i \psi_i(\underline{\boldsymbol{r}}) \qquad \underline{\boldsymbol{r}} \in \Omega$$
$$\psi_i(\underline{\boldsymbol{r}}) = 0 \qquad \underline{\boldsymbol{r}} \in \partial\Omega$$

where

$$\hat{\mathcal{V}} = \hat{\mathcal{V}}_0 + \hat{\mathcal{V}}_H + \hat{\mathcal{V}}_x$$

with

  - the electron-nuclear interaction $\hat{\mathcal{V}}_0$
  - the Hartree potential $\hat{\mathcal{V}}_H$
  - the exchange potential $\hat{\mathcal{V}}_x$

Formulation of the problem

- . . . can be discretised on a FE grid to give an eigenproblem:

$$\mathbf{F}\underline{\boldsymbol{c}}^{(i)} = \varepsilon_i \mathbf{S}\underline{\boldsymbol{c}}^{(i)}$$

with

$$S_{jk} = \int_\Omega \varphi_j(\underline{\boldsymbol{r}})\varphi_k(\underline{\boldsymbol{r}})\,\mathrm{d}\underline{\boldsymbol{r}}$$
$$F_{jk} = \int_\Omega \frac{1}{2}\nabla\varphi_j(\underline{\boldsymbol{r}}) \cdot \nabla\varphi_k(\underline{\boldsymbol{r}}) + \varphi_j(\underline{\boldsymbol{r}})\,\hat{\mathcal{V}}(\underline{\boldsymbol{r}})\,\varphi_k(\underline{\boldsymbol{r}})\,\mathrm{d}\underline{\boldsymbol{r}}$$

and

- Orbital coefficient $\underline{\boldsymbol{c}}^{(i)}$
- Orbital energy ε_i

Evaluating integrals of local operators (1)

- Consider overlap as sum of cell-wise contributions:

$$S_{ij} = \sum_c S_{ij}^c = \sum_c \int_c \varphi_i(\underline{\boldsymbol{r}}) \varphi_j(\underline{\boldsymbol{r}}) \, \mathrm{d}\underline{\boldsymbol{r}}$$

- Transform S_{ij}^c onto reference cell
- Use exact Gaussian quadrature to evaluate
- $S_{ij}^c = 0$ iff φ_i and φ_j do not share support on c

$\Rightarrow$ Sparsity pattern in S_{ij} known *before* computation

$\Rightarrow$ $\mathcal{O}(N_{\mathrm{FE}})$ in space and time

Evaluating integrals of local operators (2)

- Effect of local operators like $\hat{\mathcal{V}}_0$ or Δ inside cell

$\Rightarrow$ Integrals

$$T_{ij} = \int_\Omega \frac{1}{2} \nabla\varphi_i(\underline{\boldsymbol{r}}) \cdot \nabla\varphi_j(\underline{\boldsymbol{r}}) \, \mathrm{d}\underline{\boldsymbol{r}}$$

and

$$(V_0)_{ij} = \int_\Omega \varphi_i(\underline{\boldsymbol{r}}) \hat{\mathcal{V}}_0 \varphi_j(\underline{\boldsymbol{r}}) \, \mathrm{d}\underline{\boldsymbol{r}}$$

are automatically $\mathcal{O}(N_{\mathrm{FE}})$.

Evaluating the coulomb integral

$$J_{jk} = \int_\Omega \varphi_j(\underline{\boldsymbol{r}})\, \hat{\mathcal{V}}_H(\underline{\boldsymbol{r}})\, \varphi_k(\underline{\boldsymbol{r}})\, \mathrm{d}\underline{\boldsymbol{r}}$$

$$\hat{\mathcal{V}}_H(\underline{\boldsymbol{r}}_1) = \sum_{i \in \mathrm{occ}} \int_\Omega \frac{|\psi_i(\underline{\boldsymbol{r}}_2)|^2}{r_{12}}\, \mathrm{d}\underline{\boldsymbol{r}}_2$$

- $\hat{\mathcal{V}}_H(\underline{\boldsymbol{r}})$ is a local potential
- Obtained by solving Poisson eq^{n}:

$$-\Delta \hat{\mathcal{V}}_H(\underline{\boldsymbol{r}}) = 4\pi\rho(\underline{\boldsymbol{r}}) \qquad \underline{\boldsymbol{r}} \in \Omega$$

- Can be done in $\mathcal{O}(N_{\mathrm{FE}})$
- Might need higher-order discretisation for Poisson problem.

Non-local exchange operator

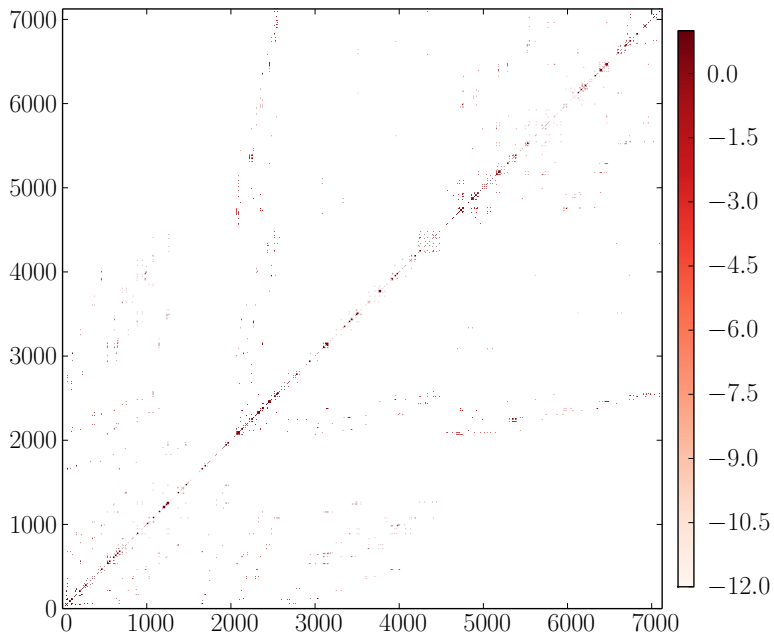
$$
\begin{aligned}
K_{jk} &= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1)\, \hat{\mathcal{V}}_x(\underline{\boldsymbol{r}}_1)\, \varphi_k(\underline{\boldsymbol{r}}_1)\, \mathrm{d}\underline{\boldsymbol{r}}_1 \\
&= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\, \mathrm{d}\underline{\boldsymbol{r}}_2\, \mathrm{d}\underline{\boldsymbol{r}}_1
\end{aligned}
$$

- $\hat{\mathcal{V}}_x(\underline{\boldsymbol{r}})$ is non-local
- $\mathcal{O}(N_{FE}^2)$ in both storage and time

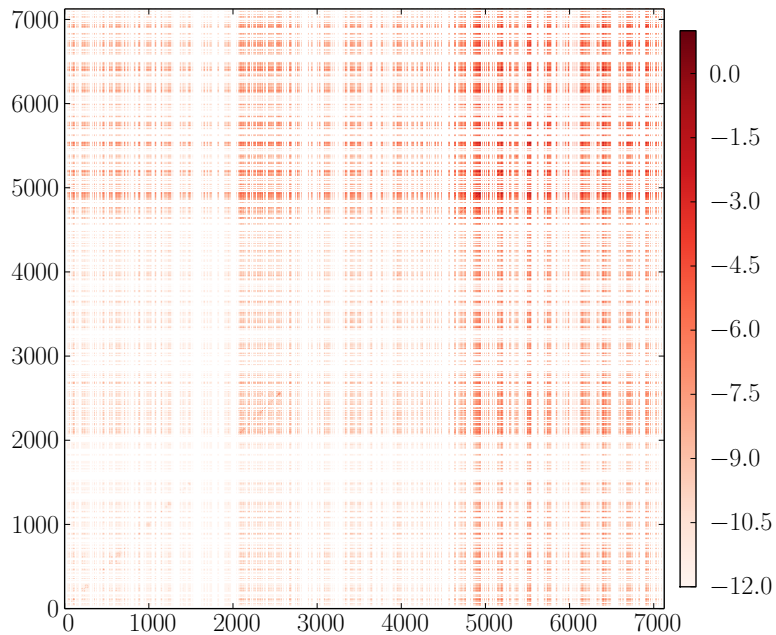
$\Rightarrow$ We cannot store $\mathbf{K}$ (or $\mathbf{F}$)

Local vs. non-local operators

$\mathbf{T} + \mathbf{V}_0 + \mathbf{J}$



$\mathbf{K}$



- Colouring depends on absolute logarithm of elements

Application of $\mathbf{K}$ is feasible

$$(\mathbf{K}\underline{\tilde{\boldsymbol{c}}})_j = \sum_k \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1\ \tilde{c}_k$$

- Application of $\mathbf{K}$ requires solving N_occ Poisson equations

$\Rightarrow$ $\mathcal{O}(N_\mathrm{occ}N_\mathrm{FE})$ in memory and computational cost

- Which inner discretisation?
- Seek methods to reduce effort per Poisson equation

Application of $\mathbf{K}$ is feasible

$$
\begin{aligned}
(\mathbf{K}\underline{\tilde{\boldsymbol{c}}})_j &= \sum_k \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1\ \tilde{c}_k \\
&= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\psi_i(\underline{\boldsymbol{r}}_2)\tilde{\psi}(\underline{\boldsymbol{r}}_2)}{r_{12}}\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1
\end{aligned}
$$

- Application of $\mathbf{K}$ requires solving N_occ Poisson equations

$\Rightarrow$ $\mathcal{O}(N_\mathrm{occ}N_\mathrm{FE})$ in memory and computational cost

- Which inner discretisation?
- Seek methods to reduce effort per Poisson equation

Application of $\mathbf{K}$ is feasible

$$
\begin{aligned}
(\mathbf{K}\underline{\tilde{\boldsymbol{c}}})_j &= \sum_k \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1\ \tilde{c}_k \\
&= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1) \underbrace{\int_\Omega \frac{\psi_i(\underline{\boldsymbol{r}}_2)\tilde{\psi}(\underline{\boldsymbol{r}}_2)}{r_{12}}\,\mathrm{d}\underline{\boldsymbol{r}}_2}\,\mathrm{d}\underline{\boldsymbol{r}}_1 \\
&= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1) \qquad \tilde{V}_i(\underline{\boldsymbol{r}}_1) \qquad \mathrm{d}\underline{\boldsymbol{r}}_1
\end{aligned}
$$

- Application of $\mathbf{K}$ requires solving N_{occ} Poisson equations

$\Rightarrow$ $\mathcal{O}(N_{\mathrm{occ}}N_{\mathrm{FE}})$ in memory and computational cost

- Which inner discretisation?
- Seek methods to reduce effort per Poisson equation

Application of $\mathbf{K}$ is feasible

$$(\mathbf{K}\underline{\tilde{\boldsymbol{c}}})_j = \sum_k \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1\ \tilde{c}_k$$
$$= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\tilde{V}_i(\underline{\boldsymbol{r}}_1)\,\mathrm{d}\underline{\boldsymbol{r}}_1$$

with

$$-\Delta\tilde{V}_i(\underline{\boldsymbol{r}}) = \psi_i(\underline{\boldsymbol{r}}) \sum_k \tilde{c}_k\varphi_k(\underline{\boldsymbol{r}})$$

- Application of $\mathbf{K}$ requires solving N_occ Poisson equations

$\Rightarrow$ $\mathcal{O}(N_\mathrm{occ}N_\mathrm{FE})$ in memory and computational cost

- Which inner discretisation?
- Seek methods to reduce effort per Poisson equation

Application of $\mathbf{K}$ is feasible

$$(\mathbf{K}\underline{\tilde{\boldsymbol{c}}})_j = \sum_k \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \int_\Omega \frac{\sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\psi_i(\underline{\boldsymbol{r}}_2)}{r_{12}} \varphi_k(\underline{\boldsymbol{r}}_2)\,\mathrm{d}\underline{\boldsymbol{r}}_2\,\mathrm{d}\underline{\boldsymbol{r}}_1\ \tilde{c}_k$$
$$= \int_\Omega \varphi_j(\underline{\boldsymbol{r}}_1) \sum_{i\in\mathrm{occ}} \psi_i(\underline{\boldsymbol{r}}_1)\tilde{V}_i(\underline{\boldsymbol{r}}_1)\,\mathrm{d}\underline{\boldsymbol{r}}_1$$

with

$$-\Delta\tilde{V}_i(\underline{\boldsymbol{r}}) = \psi_i(\underline{\boldsymbol{r}}) \sum_k \tilde{c}_k\varphi_k(\underline{\boldsymbol{r}})$$

- Application of $\mathbf{K}$ requires solving N_{occ} Poisson equations

$\Rightarrow$ $\mathcal{O}(N_{\mathrm{occ}}N_{\mathrm{FE}})$ in memory and computational cost

- Which inner discretisation?
- Seek methods to reduce effort per Poisson equation

Notes and observations (1)

- Need to treat various terms of $\mathbf{F}$ differently:
  - $\mathbf{T}$ and $\mathbf{V}_0$ can be stored
  - $\mathbf{J}$ and $\mathbf{K}$ require solution of linear systems
  - $\mathbf{K}$ can only be applied

$\Rightarrow$ Only iterative diagonalisation possible

$\Rightarrow$ Can only compute some eigenpairs of $\mathbf{F}$

Notes and observations (2)

- Obtaining exact $\mathbf{K}$ not yet successful
- Many details still unknown:
  - What eigensolver / Poisson solver algorithms work best?
  - How accurate does the FE grid need to be?
  - How about the grid for solving the Poisson equations?
  - What algorithms should be chosen for the numerical integration in $\mathbf{K}\underline{\tilde{c}}$?
  - Non-linear eigensolver possible?

$\Rightarrow$ We need a framework to try things out.

Contents

Current state of quantum chemistry software

- Large program packages
- Highly optimised towards Gaussians:
  - Optimisations in integral screening / computation
  - Very well-engineered evaluation order
- Many decisions hard-coded:
  - Dense matrix structure
  - Eigensolver
  - Linear algebra backend
- Tens of years of work: Very fast programs

Our goal (1)

- Give novel basis function types a try:
  - Sturmians
  - Finite Differences
  - Finite Elements

⇒ For fair comparison: Everything optimised to the same level

- Be able to experiment
  - Large library of eigensolvers
  - Interfaces to many linear algebra (LA) backends
  - Interfaces to high-level languages (Python, . . .)

⇒ Easy to use abstractions

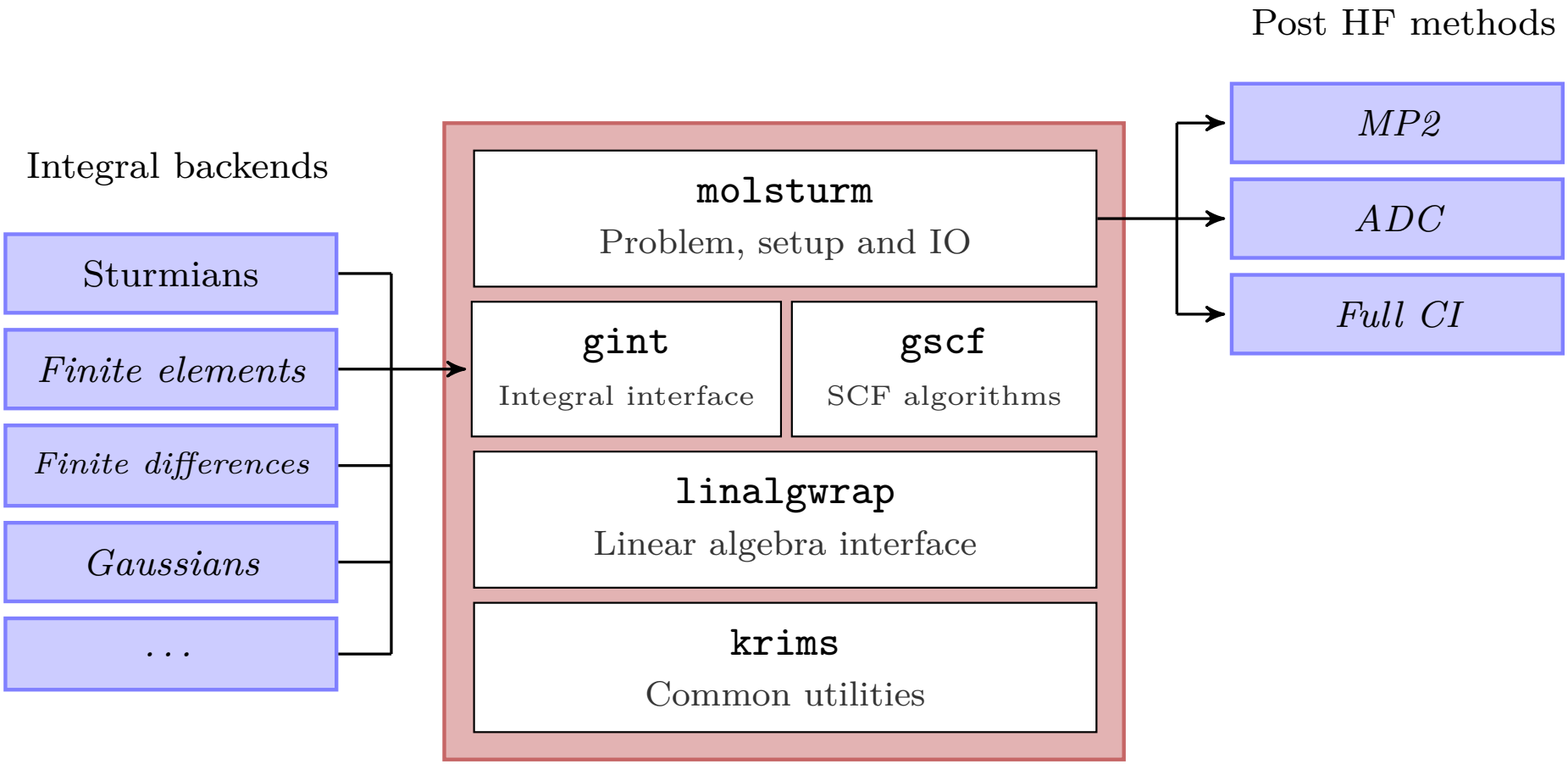
Our goal (2)

- Modular program structure, e.g.
  - SCF code should be independent of basis type
  - I/O should be independent of SCF algorithm
  - Allow to try new architectures or computing models

$\Rightarrow$ Need abstract way to express algorithms:

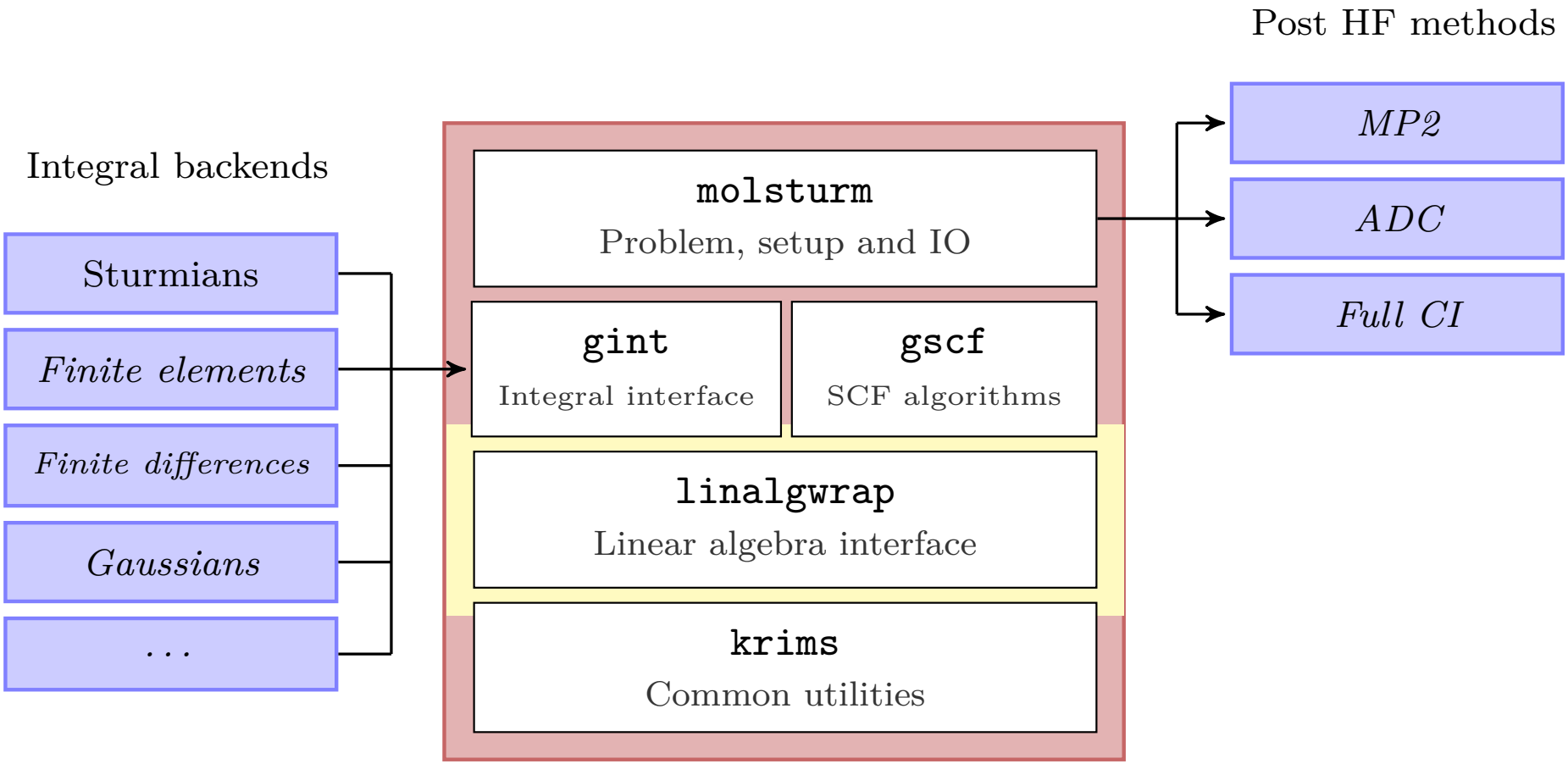
- Algorithms describe what computation should be done
- Backend decides how computation is done

molsturm overview



- *Italic*: planned

molsturm overview



Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Offers matrix-like interface
  - Obtaining elements expensive
  - *Operators*: Mainly used via matrix-vector application (`gemv`)
  - Evaluation is lazy, i.e.

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

  - On application:

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

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Offers matrix-like interface
  - Obtaining elements expensive
  - *Operators*: Mainly used via matrix-vector application (`gemv`)
  - Evaluation is lazy, i.e.

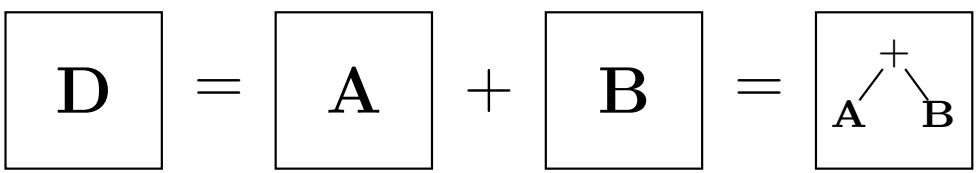
$$\boxed{\mathbf{D}} = \boxed{\mathbf{A}} + \boxed{\mathbf{B}}$$

  - On application:

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

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Offers matrix-like interface
  - Obtaining elements expensive
  - *Operators*: Mainly used via matrix-vector application (`gemv`)
  - Evaluation is lazy, i.e.

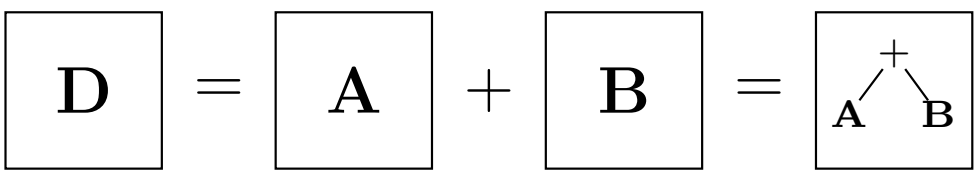


  - On application:

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

Lazy matrices

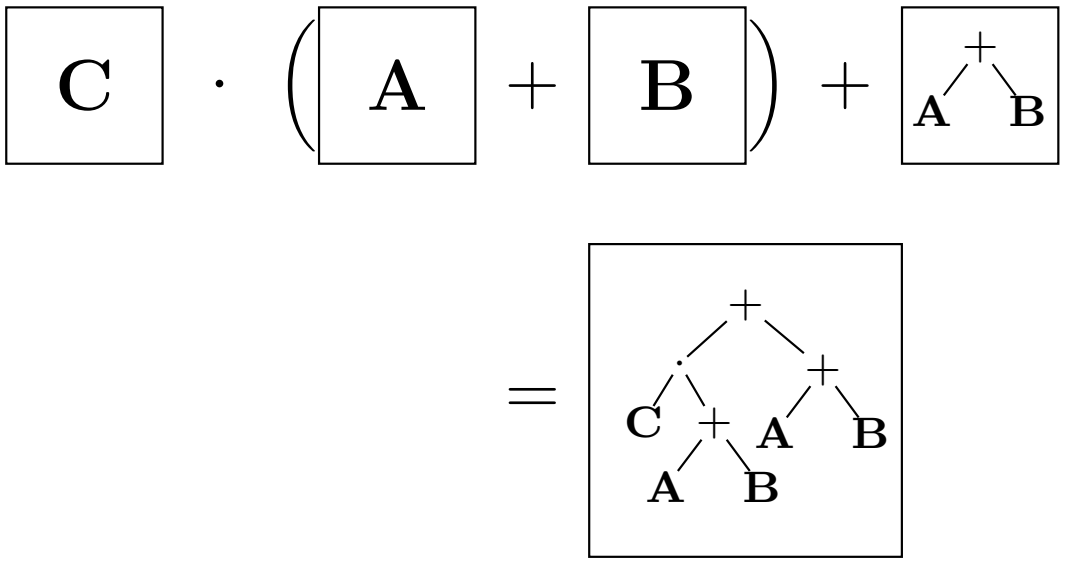
- Stored matrix: All elements reside in memory
- Lazy matrix:
  - Offers matrix-like interface
  - Obtaining elements expensive
  - *Operators*: Mainly used via matrix-vector application (`gemv`)
  - Evaluation is lazy, i.e.



  - On application:

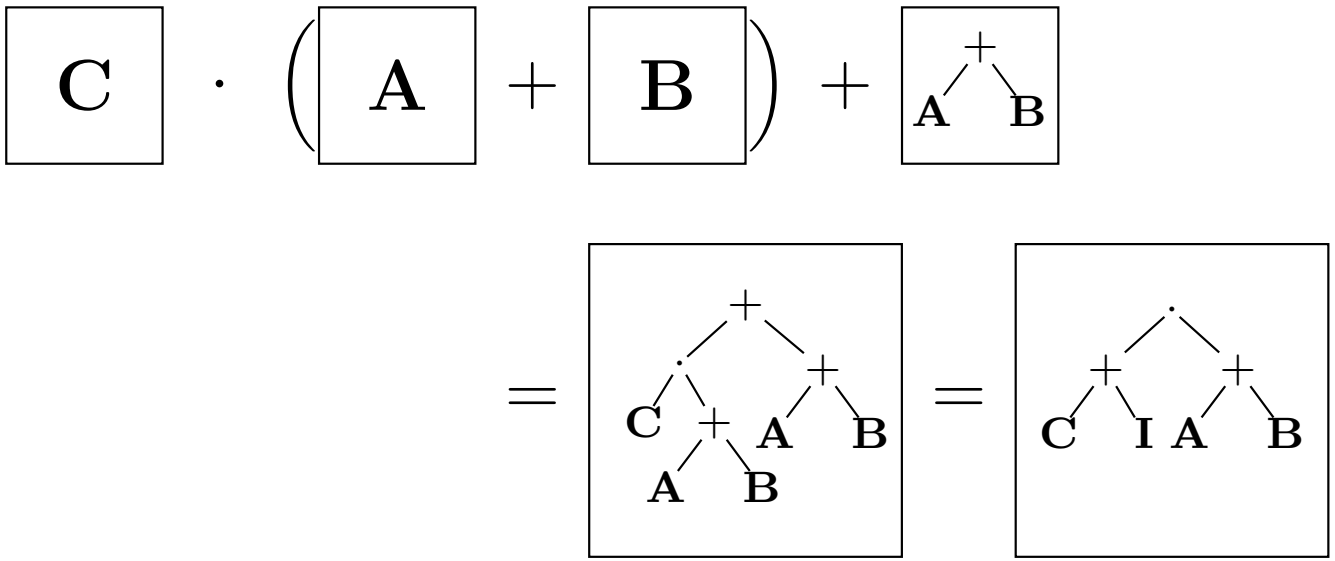
$$\mathbf{D}\underline{\boldsymbol{x}} = (\mathbf{A}\underline{\boldsymbol{x}}) + (\mathbf{B}\underline{\boldsymbol{x}})$$

A more complicated example



- Expression tree may be optimised planned
- Graph theory problem
- Methods of program analysis and optimisation

A more complicated example



- Expression tree may be optimised planned
- Graph theory problem
- Methods of program analysis and optimisation

Notes and 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 LA backend
- LA backend and tree optimisations abstracted

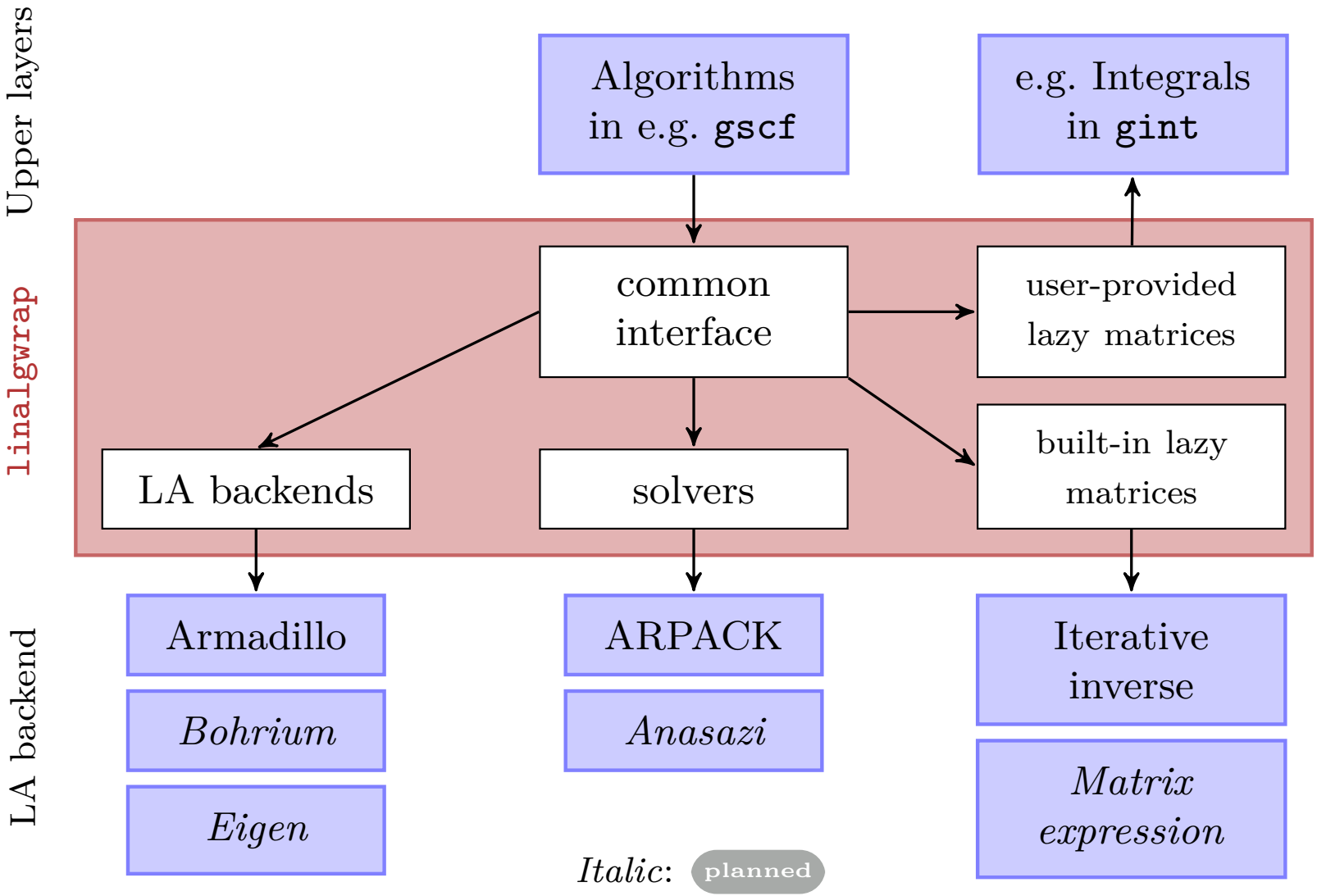
$\Rightarrow$ Language for writing `gemv`-based algorithms

$\Rightarrow$ Proper modularisation between

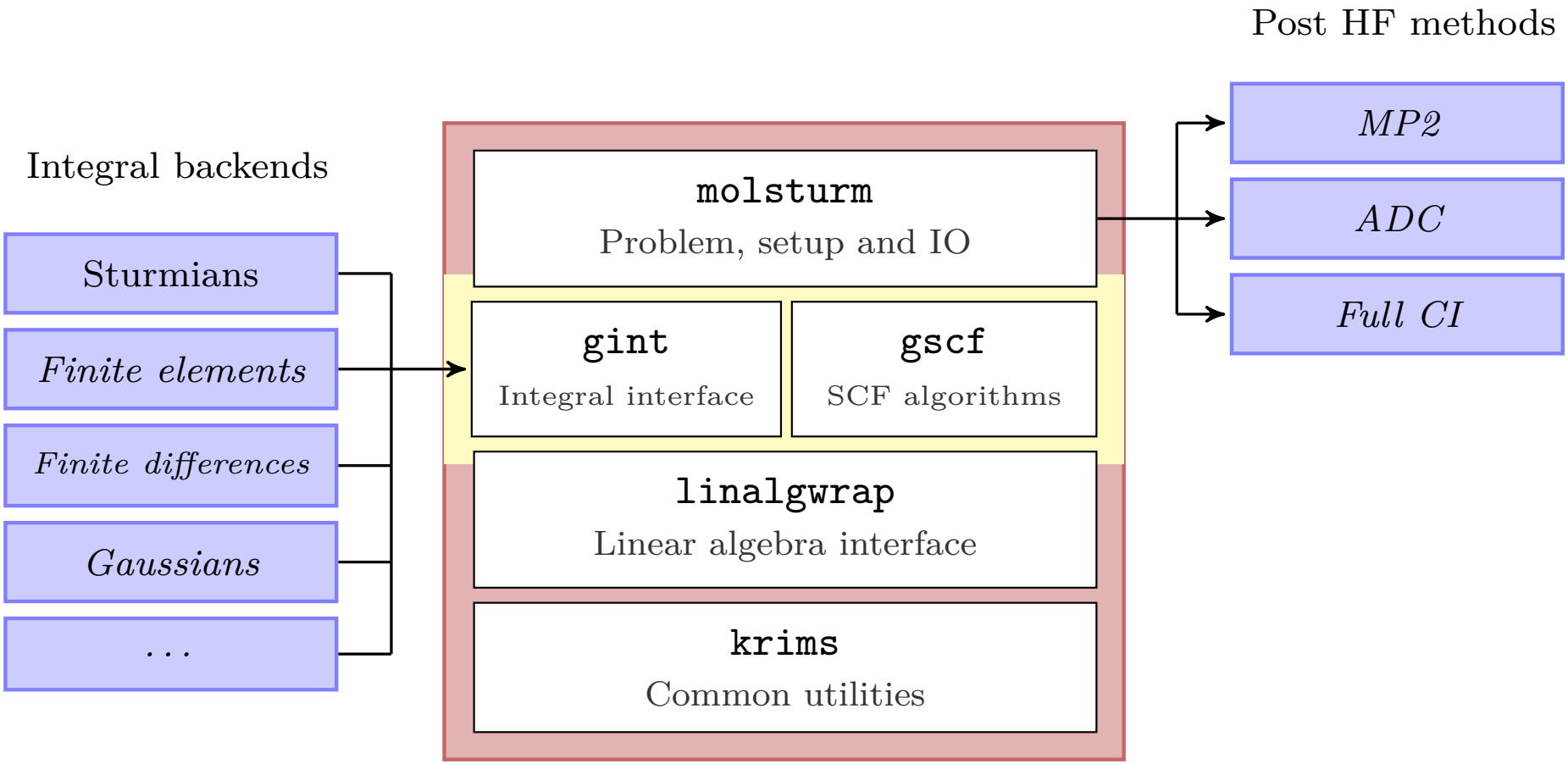
- Algorithms (i.e. `gscf`)
- Matrix implementations (i.e. `gint`)
- LA backends

linalgwrap

Linear algebra wrapper library — https://linalgwrap.org



molsturm overview



gint

Integral interface

- Integrals: Lazy matrices
- Integral backend:
  - Chooses storage scheme for matrices and vectors
  - `gemv` implementation
- `gint` contains
  - Integral selection
  - Screening planned
  - Cutoffs planned
  - Basis set projection planned

gscf

SCF algorithms

- Matrix-free: partial
  - Based on `gemv`
  - Density matrix never built up explicitly
  - Iterative eigensolvers

  $\Rightarrow$ Does not yield all virtual orbitals
- Algorithms:
  - Plain
  - DIIS
  - Optimal damping algorithm planned
- Open shell is planned
- Sensible SCF guess planned

gscf

SCF algorithms

- Matrix-free: partial
  - Based on `gemv`
  - Density matrix never built up explicitly
  - Iterative eigensolvers
  - ⇒ Does not yield all virtual orbitals
- Algorithms:
  - Plain
  - DIIS
  - Optimal damping algorithm planned
- Open shell is planned
- Sensible SCF guess planned

gscf

SCF algorithms

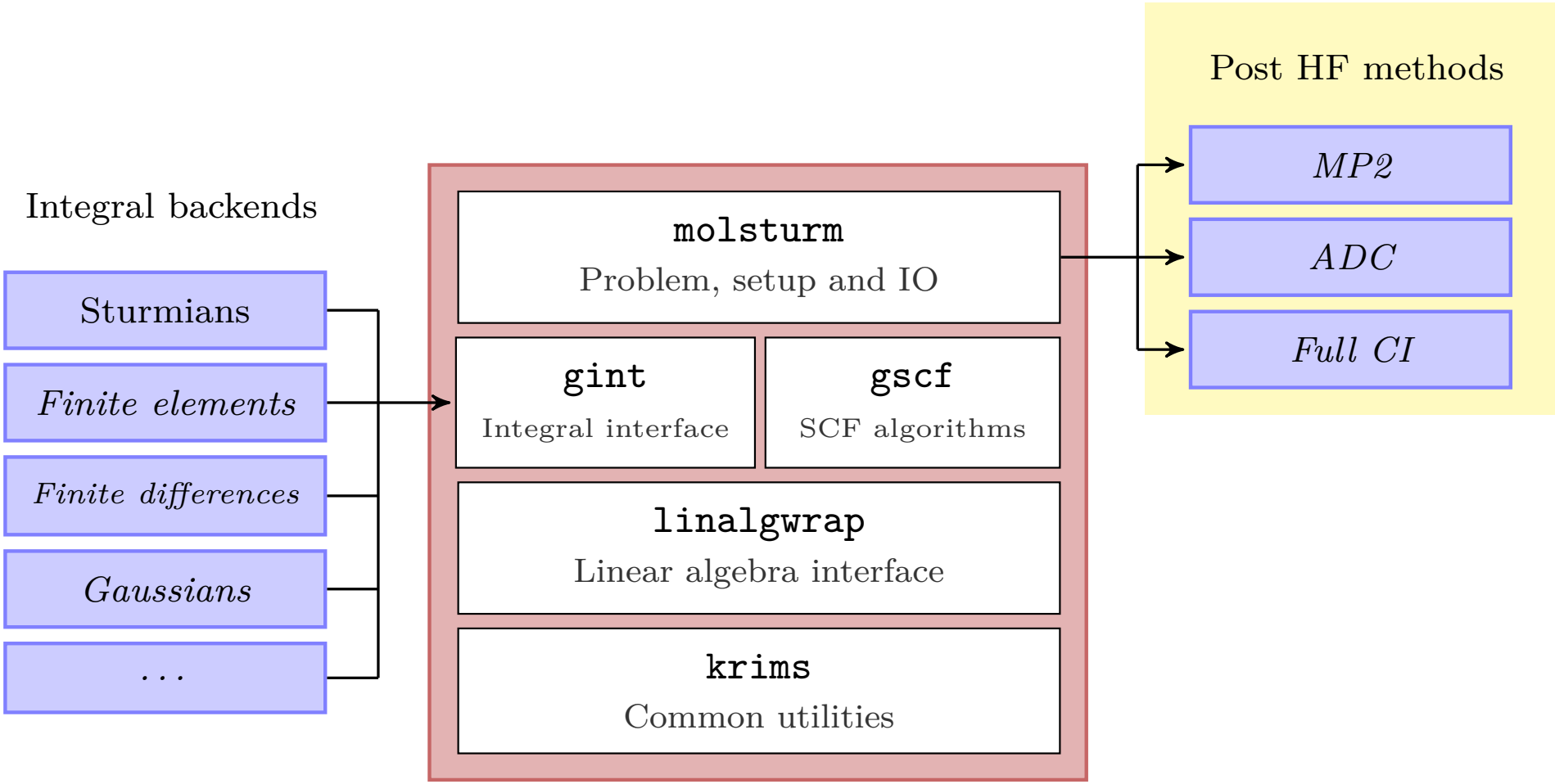
- Matrix-free: partial
  - Based on `gemv`
  - Density matrix never built up explicitly
  - Iterative eigensolvers
  - ⇒ Does not yield all virtual orbitals
- Algorithms:
  - Plain
  - DIIS
  - Optimal damping algorithm planned
- Open shell is planned
- Sensible SCF guess planned

Demo

DEMO

of a Sturmian SCF

molsturm overview



Post HF with molsturm

- molsturm may export
  - Fock matrix (MO basis)
  - Two electron integrals (MO basis)
  - Orbital coefficients
  - Orbital energies
- Possible Post-HF methods: planned
  - ADC
  - (Matrix-free) FCI
  - Python interface

Contents

Summary

- Finite Elements:
  - Flexible numerical basis
  - Unusual requirements
  - Need `gemv` based SCF
- `molsturm`
  - Modular Hartree-Fock framework
  - Designed for experimentation
  - Based on `gemv` operations
  - Free software (partial: `https://linalgwrap.org`)

Outlook

The near future

- Matrix-free Sturmian SCF
- Gaussian basis functions
- ADC interface
- Finite difference based quantum chemistry
- Interface to more Eigensolvers (Anasazi)

Acknowledgements

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

- Prof. Guido Kanschat
- HGS Mathcomp

References

- P. Bastian, *Scientific Computing with Partial Differential Equations*. Lecture notes, Ruprecht-Karls-Universität Heidelberg, 2014.
- J. Avery, *New Computational Methods in the Quantum Theory of Nano-Structures*. PhD thesis, University of Copenhagen, 2011.
- J. Avery and J. Avery, *Generalized Sturmians and Atomic Spectra*. World Scientific, 2006.
- `https://linalgwrap.org`