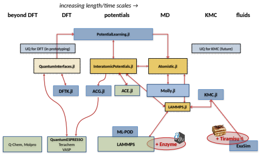
Fostering interdisciplinary research by composable julia software

Michael F. Herbst

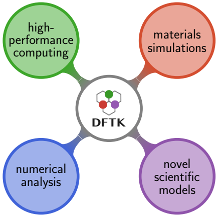
Mathematics for Materials Modelling (`matmat.org`), EPFL

20 February 2024

Slides: `https://michael-herbst.com/talks/2024.02.20_elstruct_code_workshop.pdf`



Real-world multi-physics
software stack for materials modelling



$$\text{Stress} = \frac{1}{\det(\mathbf{L})} \left. \frac{\partial E\left[P_*, (I+\mathbf{M})\,\mathbf{L}\right]}{\partial \mathbf{M}} \right|_{\mathbf{M}=0}$$

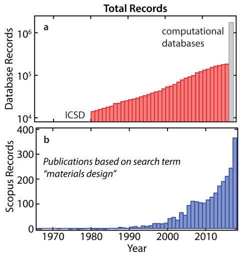
```
# Run SCF, get P*
scfres = self_consistent_field(basis)
     L = basis.model.lattice
stress = 1/det(L) * gradient(
    M -> recompute_energy(
                 scfres, (I + M) * L),
    zero(L)
)
```

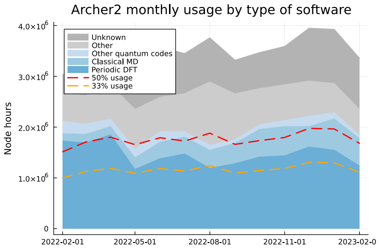
julia vision: Math $\equiv$ code

Tackling 21st century challenges

- 21st century challenges:
  - Renewable energy, green chemistry, health care . . .
- Current solutions limited by properties of available materials

  $\Rightarrow$ Innovation driven by discovering new materials
- Crucial tool: Computational materials discovery
  - Systematic simulations on $\simeq 10^4 - 10^6$ compounds
  - Complemented by data-driven approaches
  - Noteworthy share of world's supercomputing resources





Tackling 21st century challenges

- 21st century challenges:
  - Renewable energy, green chemistry, health care . . .
- Current solutions limited by properties of available materials

  $\Rightarrow$ Innovation driven by discovering new materials
- Crucial tool: Computational materials discovery
  - Systematic simulations on $\simeq 10^4 - 10^6$ compounds
  - Complemented by data-driven approaches
  - Noteworthy share of world's supercomputing resources
- Multi-disciplinary effort: Software takes a key role
  - E.g. growing list of data / workflow management tools
  - Challenges of combining efforts & integrating communities

A focus on robust materials simulations

- **Goal** in MatMat group:
  - Obtain reliable & efficient simulations
  - Develop and employ mathematical analysis of error
  - Transform empirical wisdom to built-in convergence guarantees

$\Rightarrow$ Understand where and how to spend efforts best

- Practical error indicators:
  - Automatic & robust verification
  - Multi-fidelity statistical surrogates
  - Active learning of missing physics
- Leverage inexactness:
  - Error balancing: Optimal adaptive parameter selection
  - Adaptive tolerances & selective precision

$\Rightarrow$ Multidisciplinary expertise required

A focus on robust materials simulations

- **Goal** in MatMat group:
  - Obtain reliable & efficient simulations
  - Develop and employ mathematical analysis of error
  - Transform empirical wisdom to built-in convergence guarantees

$\Rightarrow$ Understand where and how to spend efforts best

- Practical error indicators:
  - Automatic & robust verification
  - Multi-fidelity statistical surrogates
  - Active learning of missing physics

- Leverage inexactness:
  - Error balancing: Optimal adaptive parameter selection
  - Adaptive tolerances & selective precision

$\Rightarrow$ Multidisciplinary expertise required

(Exaggerative) state of codes in this field

Mathematical research

- **Goal:** Numerical experiments
- **Scope:** Reduced models
- High-level **language**: Matlab, python, . . .
- **Lifetime:** 1 paper
- **Size:** < 1k lines
- Does not care about performance

Application research

- **Goal:** Modelling physics
- **Scope:** All relevant systems
- Mix of **languages:** C, FORTRAN, python, . . .
- **Lifetime:** 100 manyears
- **Size:** 100k – 1M lines
- Obliged to write performant code

- Working with these codes requires different skillsets
  - $\Rightarrow$ Orthogonal developer & user communities
- Obstacle for knowledge transfer:
  - Mathematical methods never tried in practical setting (and may well not work well in the real world)
  - Some issues cannot be studied with mathematical codes (and mathematicians may never get to know of them)
- What about emerging hardware, accelerators, performance?
  - Should be the regime of Computer Science (yet another community)

Difficulties of interdisciplinary research

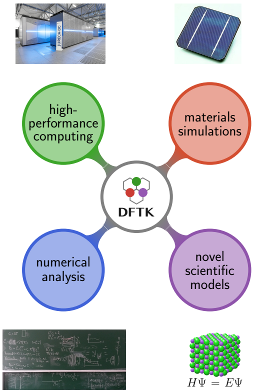
- Community conventions (e.g. publication culture)
- Language barriers and context-sensitive terms
- Speed of research (development of model vs. its analysis)

- A social problem . . .
  - (Communication, convention, compromises, . . .)

- . . . that is cemented in software:
  - Priorities differ $\Rightarrow$ What is considered “a good code” differs
  - Insurmountable obstacles to integrate codes
  - Collaborations can stop before they begin . . .

- **Hypothesis:** People compose if software composes

Density-functional toolkit (DFTK) — `https://dftk.org`



- julia-based DFT code in 7500 lines
- Cross-community: Mathematical research & applications
- Allows restriction to relevant model problems,
- *and* scale-up to application regime (1000 electrons)
- Integrated with high-throughput: MARVEL AiiDA
- Lessons learned:
  - Software integration is hard work
  - Unexpected catalytic effects from integration discussions
  - Parties understand their role, change of viewpoint
  - $\Rightarrow$ As software composes, communities compose
- **Central:** How can we lower the barrier to integrate?

We already want a lot from good software . . .

- Integration across communities (users, developers, scientists)
- Maintainability
- Reproducibility
- Documentation / Accessibility
- Portability (future technologies & hardware)
- Performance
- . . .

- Can we the get the best in each category?
  - Probably not . . .

- To maximise integration: Where should we compromise?

Contents

Separating the what from the how

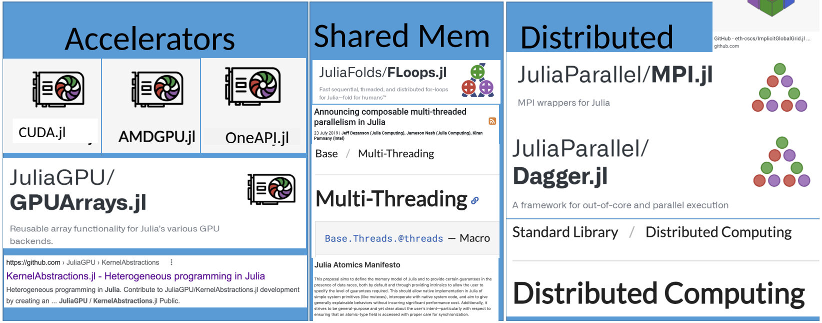
- Why is this separation so important . . .
  - . . . for composable software?
  - . . . for multidisciplinary research?

- Consider the **goal**: Modelling a physical system
- Traditionally users code in detail how the computation should proceed (Imperative programming)
  - How = architecture
  - How = algorithm
  - How = memory layout
  - How = discretisation
  - . . .

- But all this has nothing to do with physics!
- Can the how be abstracted away?
  - such that CS / Math can deal with it *independently*

- Let's see some julia developments

HPC abstractions



Accelerators

CUDA.jl

AMDGPU.jl

OneAPI.jl

JuliaGPU/**GPUArrays.jl**

Reusable array functionality for Julia's various GPU backends.

https://github.com › JuliaGPU › KernelAbstractions

KernelAbstractions.jl - Heterogeneous programming in Julia

Heterogeneous programming in **Julia**. Contribute to JuliaGPU/KernelAbstractions.jl development by creating an ... **JuliaGPU** / **KernelAbstractions**.jl Public.

Shared Mem

JuliaFolds/**FLoops.jl**

Fast sequential, threaded, and distributed for-loops for Julia—fold for humans™

Announcing composable multi-threaded parallelism in Julia

23 July 2019 | **Jeff Bezanson (Julia Computing), Jameson Nash (Julia Computing), Kiran Pamnany (Intel)**

Base / Multi-Threading

Multi-Threading

`Base.Threads.@threads` — Macro

Julia Atomics Manifesto

This proposal aims to define the memory model of Julia and to provide certain guarantees in the presence of data races, both by default and through providing intrinsics to allow the user to specify the level of guarantees required. This should allow native implementation in Julia of simple system primitives (like mutexes), interoperate with native system code, and aim to give generally explainable behaviors without incurring significant performance cost. Additionally, it strives to be general-purpose and yet clear about the user's intent—particularly with respect to ensuring that an atomic-type field is accessed with proper care for synchronization.

Distributed

JuliaParallel/**MPI.jl**

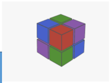
MPI wrappers for Julia

JuliaParallel/**Dagger.jl**

A framework for out-of-core and parallel execution

Standard Library / Distributed Computing

Distributed Computing



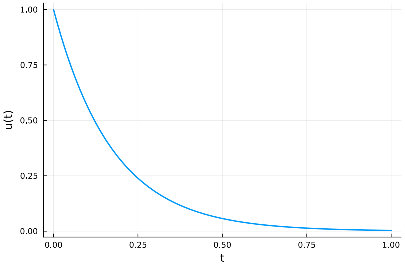
GitHub - eth-cscs/ImplicitGlobalGrid.jl ...
github.com

```julia
function power_method(A, x; niter=100)
    for i = 1:niter
        x = A * x
        x ./= norm(x)
    end
    x
end
```

```julia
A = rand(10, 10); A = A + A' + 10I; x = rand(10)

using LinearMaps, IterativeSolvers
itinv(A) = LinearMap(x -> cg(A, x), size(A)...)

using CUDA
power_method(itinv(CuArray(A)), CuArray(x))

using AMDGPU
power_method(itinv(ROCArray(A)), ROCArray(x))
```

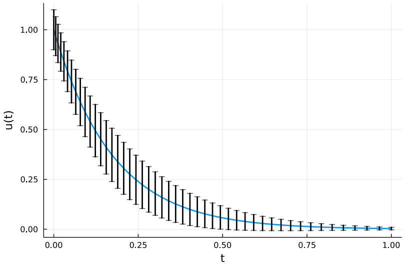
Code reinterpretation & self-implementing features

```
using OrdinaryDiffEq, Plots

# Half-life of Carbon-14 is 5730 years.
c = 5.730

# Setup
u0 = 1.0
tspan = (0.0, 1.0)

# Define the problem
radioactivedecay(u, p, t) = -c*u

# Pass to solver
prob = ODEProblem(radioactivedecay, u0, tspan)
sol = solve(prob, Tsit5();
            reltol=1e-8, abstol=1e-8)

plot(sol.t, sol.u;
     ylabel="u(t)", xlabel="t", lw=2, legend=false)
```



Code reinterpretation & self-implementing features

```
using OrdinaryDiffEq, Measurements, Plots

# Half-life of Carbon-14 is 5730 years.
c = 5.730 ± 2

# Setup
u0 = 1.0 ± 0.1
tspan = (0.0, 1.0)

# Define the problem
radioactivedecay(u, p, t) = -c*u

# Pass to solver
prob = ODEProblem(radioactivedecay, u0, tspan)
sol = solve(prob, Tsit5();
            reltol=1e-8, abstol=1e-8)

plot(sol.t, sol.u;
     ylabel="u(t)", xlabel="t", lw=2, legend=false)
```



- User says: I want to track measurement error
- Numerics adapts, plotting adapts
  - No prior discussion with/amongst package maintainers to "make this happen"

- `Measurement.jl` reinterprets floating-point operations
  - In some sense this feature "implemented itself"

Aside: julia package manager and binary dependencies

- julia makes no compromises in reproducibility
  - Package environments automatically tracked in plain-text files
    (Can be committed along code)
  - Includes python & foreign-code binaries

- Difficulty: Integration with HPC clusters:
  - E.g. making use of vendor-specific MPI / BLAS libraries
  - julia solution: *trampoline* libraries
  - BLAS & MPI libraries can be switched *at runtime*

  $\Rightarrow$ Sane defaults for laptops & flexibility

- Some pain points remain:
  - Default binaries cannot make full use of hardware
  - Automatic detection of vendor libraries

julia and composable software

- Magic of julia:
  - Painless generics and abstractions
  - Enables unusual code *reinterpretation*
    (Algorithmic differentiation, symbolics, cross-platform compilation)

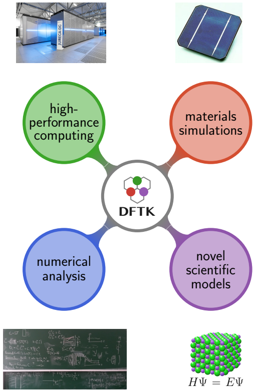
$\Rightarrow$ Separation of what and how:
  - Hardware & architecture (Computer Science)
  - Algorithms (Mathematics)
  - Model building (Physics)
  - Interactive scripting (Application scientists)

$\Rightarrow$ Cross-disciplinary expertise can compose in one code

- Modelling and algorithm code stays high-level
  - Appropriate specialisations unlock performance
  - We can add them gradually as needed (Iterative optimisation)

Contents

Density-functional toolkit[1] — `https://dftk.org`



- julia code for cross-disciplinary research:
  - Allows restriction to relevant model problems,
  - *and* scale-up to application regime (1000 electrons)
  - Sizeable feature set in 7500 lines of code
  - Including some unique features (Self-adapting algorithms)
  - Integrated with high-throughput: MARVEL AiiDA
- Fully composable due to julia abstractions:
  - Arbitrary precision (32bit, >64bit, . . .)
  - Algorithmic differentiation (AD)
  - HPC tools: GPU acceleration, MPI parallelisation
- Accessible high-productivity research framework:
  - Key contributions by undergrads (AD, GPU, Pseudos, . . .)
  - Over 30 contributors in 5 years (Maths, physics, CS, . . .)

DFTK design: Keeping code concise & accessible

$$\text{Stress} = \frac{1}{\det(\mathbf{L})} \left. \frac{\partial E\left[P_*, (I + \mathbf{M})\,\mathbf{L}\right]}{\partial \mathbf{M}} \right|_{\mathbf{M}=0}$$

```
# Run SCF, get P*
scfres = self_consistent_field(basis)
     L = basis.model.lattice
stress = 1/det(L) * gradient(
    M -> recompute_energy(
                 scfres, (I + M) * L),
    zero(L)
)
```

- Stress computation (Definition *vs.* julia code)[1]
- Post-processing step ⇒ Not performance critical

- Comparison of implementation complexity:
  - DFTK : 20 lines[1] (forward-mode algorithmic differentiation)
  - Quantum-Espresso: 1700 lines[2]
  - ≃ 10-week GSoC project

⇒ No performance impact & accessible code

[1] https://github.com/JuliaMolSim/DFTK.jl/blob/master/src/postprocess/stresses.jl

[2] https://github.com/QEF/q-e/blob/develop/PW/src

New features from generic code: Sensitivity analysis

```
function dft_energy(a, θ)
    model = Model(a, PbeExchange(θ), ...)
    scf(model).energies.total
end
optimise_lattice(θ) = optimise(a -> dft_energy(a, θ))

sensitivities =
    ForwardDiff.gradient(optimise_lattice, θ)
```

$$a_* = \arg\min_a \mathcal{E}(a, \theta)$$

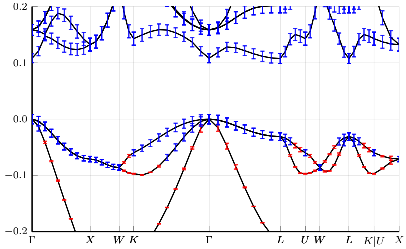
$$\text{sensitivities} = \frac{da_*}{d\theta}$$

- Arbitrary, user-desired derivatives in one line of code
  - Breaks "one PhD student per derivative" paradigm
  - $\Rightarrow$ New properties/derivatives by non-DFT experts!
- Avoids combinatorial explosion
  - Unusual derivatives equally supported

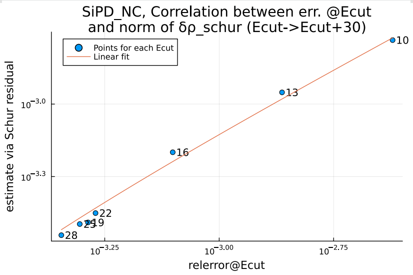
$\Rightarrow$ Setting the scene for new approaches:

- Sensitivity analysis & UQ
- Combined analytical and statistical error estimation

Support of *a posteriori* error analysis



Band structure with guaranteed errors[1]



Estimation of basis set error in ρ

- Momentum towards error estimators for DFT
  - Focus on basis set error (some also tackle floating-point, SCF convergence)
  - Estimate numerical error for modelled system (e.g. for density and forces[2])
- Results promising, but many challenges & caveats remain
  - Crucial to play with simplifications / numerics etc.

$\Rightarrow$ **DFTK** is major research tool for this development[1-4]

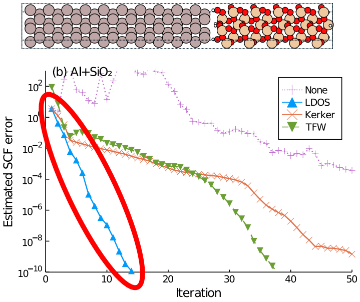
[1] MFH, A. Levitt, E. Cancès. Faraday Discus. **223**, 227 (2020).

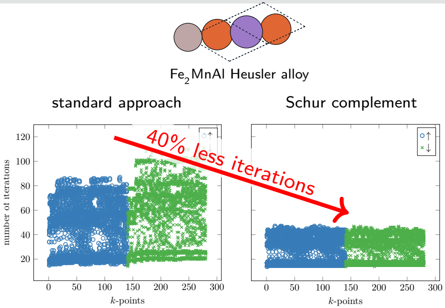
[2] E. Cancès, G. Dusson, G. Kemlin *et. al.* SIAM J. Sci. Comp., **44**, B1312 (2022).

[3] E. Cancès, G. Kemlin, A. Levitt. J. Matrix Anal. Appl., **42**, 243 (2021).

[4] E. Cancès, G. Kemlin, A. Levitt. J. Sci. Comput., **98**, 25 (2024)

Robust & efficient algorithms





- LDOS mixing for inhomogeneous systems[1] (surfaces, clusters, . . .)
- ca. 50% less iterations
- Automatic & system-adapted selection of damping[2]

- First-principle properties of metals
- Schur-complement approach to perturbation theory[2]
- ca. 40% less iterations

$\Rightarrow$ Maths / physics collaboration:
Exchange of ideas between simplified & practical settings crucial

[1] MFH, A. Levitt. J. Phys. Condens. Matter **33**, 085503 (2021).

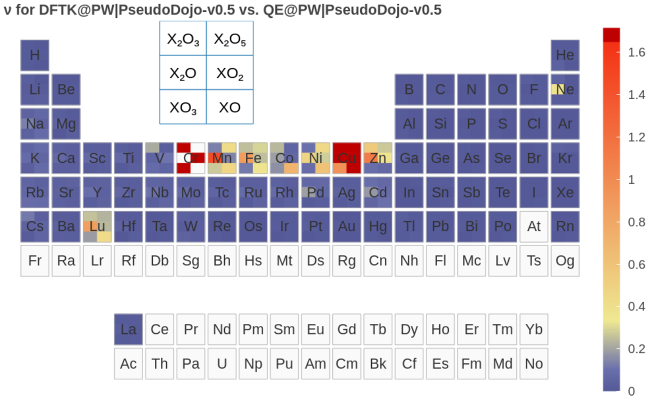
[2] MFH, A. Levitt. J. Comput. Phys. **459**, 111127 (2022).

[3] E. Cancès, MFH, G. Kemlin, *et. al.* Lett. Math. Phys. **113**, 21 (2023).

Integration with AiiDA

- Integration with AiiDA high-throughput workflow manager
  - `https://github.com/aiidaplugins/aiida-dftk`

- Used in automated verification tests: $(r_{\text{cut}} = \infty)$

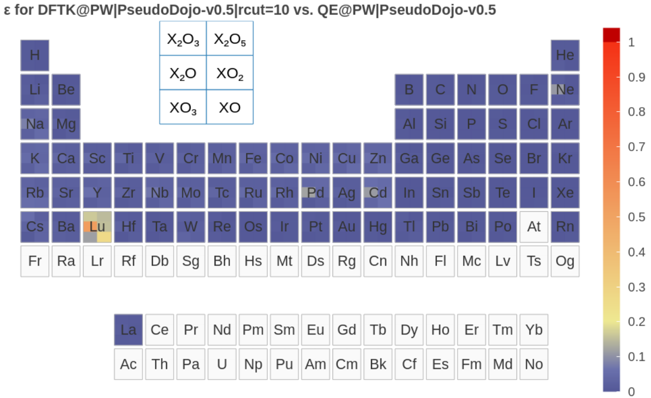


$\Rightarrow$ Excellent agreement Quantum-Espresso vs. DFTK

Integration with AiiDA

- Integration with AiiDA high-throughput workflow manager
  - `https://github.com/aiidaplugins/aiida-dftk`

- Used in automated verification tests: $(r_{\text{cut}} = 10\text{a.u.})$



$\Rightarrow$ Excellent agreement Quantum-Espresso vs. DFTK

Quick overview of julia materials codes

- Some julia materials science codes:
  - `github.com/ACEsuit`: Atomic Cluster Expansion (ML potential)
  - `JuliaMolSim/Molly.jl`: Molecular dynamics
  - `qiaojunfeng/Wannier.jl`: Wannierisation
  - `JuliaMolSim/DFTK.jl`: Density-functional theory

- Community desire for common interfaces across julia materials ecosystem
  - `AtomsBase.jl` & `AtomsCalculators.jl`
  - E.g. compatibility of structural representations *across codes*

  $\Rightarrow$ Link *within* julia ecosystem, but also *link to external codes* (e.g. ASE, Quantum Espresso, LAMMPS)

$\Rightarrow$ Generic & re-usable utility packages:
  - `AtomsIO.jl`: File parsing
  - `AtomsView.jl`: Structure viewing
  - Many just slim bindings to existing foreign-language codes ...

- Overview talk: Julia for Materials Modelling: `https://michael-herbst.com/julia-for-materials` (youtube recording)

Summary

- People compose if software composes
    - Key ingredient: Separating what and how
    - $\Rightarrow$ Better collaboration by separation of concern

- What makes julia codes so composable?
    - Specialisation: Performance & hardware specifics
    - Abstraction: Code becomes the math
    - Multiple dispatch: Repurpose existing code (e.g. AD)

- julia-based materials codes: Bridging communities
    - Multiple cross-disciplinary projects: Maths $\leftrightarrow$ applications
    - Community emphasis on composable interfaces
      (e.g. `AtomsBase.jl` & `AtomsCalculators.jl`)
    - Details: `michael-herbst.com/julia-for-materials`

Acknowledgements

- Alan Edelman (MIT)
- Valentin Churavy (MIT)
- Antoine Levitt (Université Paris-Saclay)
- All DFTK contributors
- Joe Greener (Cambridge)
- Rachel Kurchin (CMU)
- Christoph Ortner (UBC)
- Spencer Wyatt (MIT)
- Pablo Zubieta (Chicago)

EPFL MatMat Summer of code

Fonds national suisse MARVEL NATIONAL CENTRE OF COMPETENCE IN RESEARCH MIT CESMIX

Questions?

M t Mat https://matmat.org

mfherbst

michael.herbst@epfl.ch

https://michael-herbst.com/talks/2024.02.20_elstruct_code_workshop.pdf

DFTK https://dftk.org

julia https://michael-herbst.com/julia-for-materials

EPFL M t Mat