

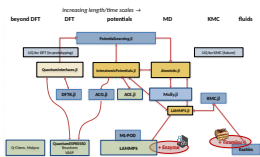
Fostering interdisciplinary research by composable **julia** software

Michael F. Herbst

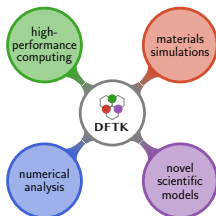
Mathematics for Materials Modelling (matmat.org), EPFL

27 June 2023

Slides: <https://michael-herbst.com/slides/pasc23>



Real-world multi-physics
software stack for materials modelling



Stress =

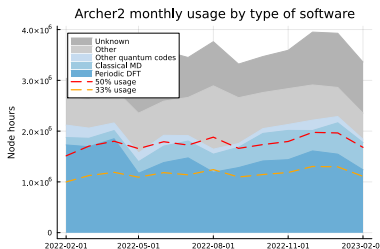
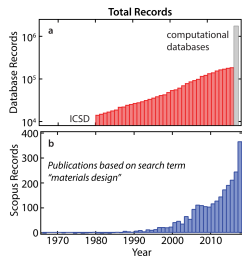
$$\frac{1}{\det(\mathbf{L})} \left. \frac{\partial E[P_*, (\mathbf{I} + \mathbf{M}) \mathbf{L}]}{\partial \mathbf{M}} \right|_{\mathbf{M}=0}$$

```
# 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
 - ⇒ 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
 - ⇒ 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



Interdisciplinary Challenges in Multiscale Materials Modeling

... and the role of software in overcoming them

This talk Composable software to integrate communities

Giovanni Pizzi Community infrastructures for high-throughput materials discovery

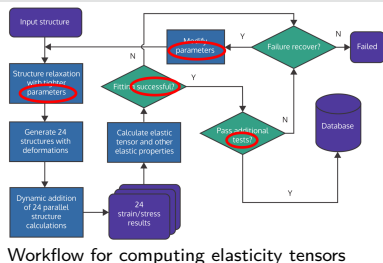
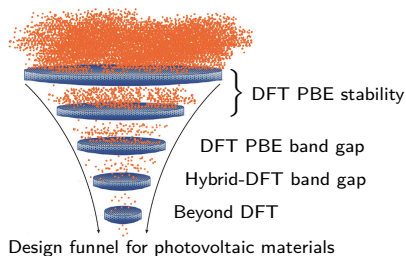
Rachel Kurchin Data-driven methods to bridge between theory and experiment

Jessica Nash Teaching and educational efforts to strengthen a software community

Contents

- 1 Challenges of integrating communities
- 2 Composability aspects of  julia
- 3 Showcases of  DFTK and related  julia efforts

Sketch of high-throughput workflows



- Many parameters to choose (algorithms, tolerances, models)
 - Elaborate heuristics: **Failure rate** $\simeq 1\%$
 - Still: **Thousands** of failed calculations
 - ⇒ **Wasted resources** & increased human attention (limits throughput)
- **Goal** in **MtMat** group: **Self-adapting black-box algorithms**
 - Transform **empirical wisdom** to built-in **convergence guarantees**
 - Requires: Uncertainty quantification & error estimation
 - ⇒ Understand **where and how** to spend efforts best

(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

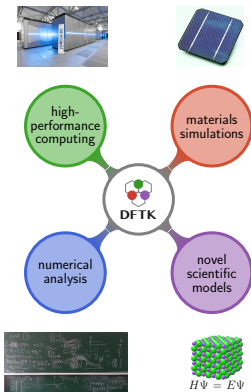
Application research

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

- Working with these codes requires different skillsets
 - ⇒ **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** ⇒ What is considered “a good code” differs
 - Insurmountable obstacles to integrate codes
 - Collaborations can stop before they begin . . .
- **Hypothesis: People compose if software composes**



- **julia**-based density-functional theory code
- Cross-community: Mathematical research & applications
- Allows restriction to **relevant model problems**,
- **and scale-up** to application regime (1000 electrons)
- Integration with multi-scale pipelines:

MARVEL



AiiDA



Mit



CESMIX

<https://nccr-marvel.ch>

<https://cesmix.mit.edu>

- **Lessons learned:**
 - Software integration is **hard work**
 - **Unexpected catalytic effects** from integration discussions
 - Each party better understands their role
- ⇒ As software composes, communities compose
- **Central goal:** How can we **lower the barrier to integrate?**


What would it take to make software integration easier?

- **Societal aspect:** We need a large open-source community
 - Fosters maintainability, reproducibility, documentation, portability, integration
- Necessary ingredients: **Change of research culture**
 - Publishing papers is not be the primary
 - Performance numbers are not be the primary
 - **Writing composable software** is the primary
- **Technical aspect:** Separating the **what** from the **how**
 - Naturally leads to separation of concern
 - ⇒ Need programming language to support this

Contents

- 1 Challenges of integrating communities
- 2 Composability aspects of  julia
- 3 Showcases of  DFTK and related  julia efforts

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

Code reinterpretation & self-implementing features

```
using OrdinaryDiffEq, Plots

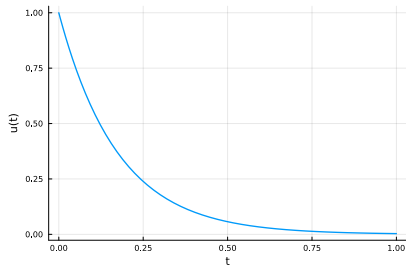
# Half-life of Carbon-14 is 5730 years.
c = 5.730

# 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

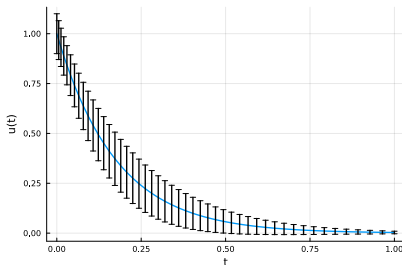
# Half-life of Carbon-14 is 5730 years.
c = 5.730 ± 2

# 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”

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

⇒ Separation of **what** and **how**:

- Hardware & architecture (Computer Science)
- Algorithms (Mathematics)
- Model building (Physics)
- Interactive scripting (Application scientists)

⇒ 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)

- **Minisymposium tomorrow** (MS5B / MS6B):

Julia for HPC: Tooling and Applications

- 1 Challenges of integrating communities
- 2 Composability aspects of  julia
- 3 Showcases of  DFTK and related  julia efforts

Density-functional theory in one slide

- **Goal:** Understand electronic structures (Many-body quantum system)
- **DFT approximation:** Effective single-particle model

$$\left\{ \begin{array}{l} \forall i \in 1 \dots N : \left(-\frac{1}{2}\Delta + V(\rho_{\Phi}) \right) \psi_i = \varepsilon_i \psi_i, \\ V(\rho) = V_{\text{nuc}} + v_C \rho + V_{\text{XC}}(\rho), \\ \rho_{\Phi} = \sum_{i=1}^N |\psi_i|^2, \\ \Phi = (\psi_1, \dots, \psi_N) \in \left(L^2(\mathbb{R}^3, \mathbb{C}) \right)^N \text{ orthogonal} \end{array} \right.$$

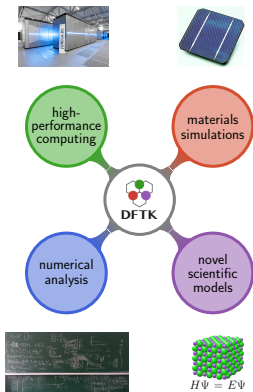
nuclear attraction V_{nuc} , exchange-correlation V_{XC} , Hartree potential $-\Delta(v_C \rho) = 4\pi\rho$

- Periodic boundary conditions & plane-wave discretisations
- **Self-consistent field (SCF):** Fixed-point problem $F(\rho) = \rho$, solved:

$$\rho_{n+1} = \rho_n + \alpha P^{-1} [F(\rho_n) - \rho_n]$$

- Hits plenty of “non-“s: Non-convex, non-linear, non-local, non-smooth

Density-functional toolkit¹ — <https://dftk.org>





- **Julia** code for plane-wave DFT, started in 2019
- **Fully composable** due to **Julia** abstractions:
 - Arbitrary precision (32bit, >64bit, ...)
 - Algorithmic differentiation (AD)
 - HPC tools: GPU acceleration, MPI parallelisation
- Low barriers 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)
- Accessible **high-productivity** research framework:
 - Key code contributions by undergrads / PhD students
 - AD support in 10 weeks (CS Bachelor)
 - GPU support in 10 weeks (Physics Bachelor)
 - Relevant contributions from outside collab. circle



Stress =

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

```
# 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)
)
```

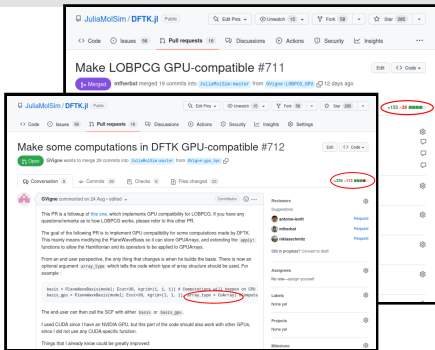
- Stress computation (Definition vs.  code)¹
- Post-processing step \Rightarrow Not performance critical
- Comparison of implementation complexity:
 -  DFTK: 20 lines¹ (forward-mode algorithmic differentiation)
 - Quantum-Espresso: 1700 lines²
 - \simeq 10-week GSoC project

\Rightarrow 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>

GPU support in DFTK



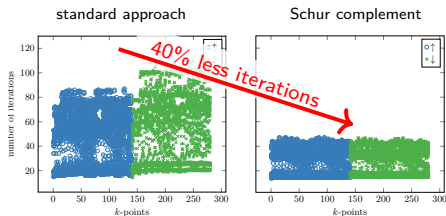
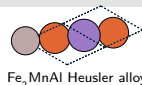
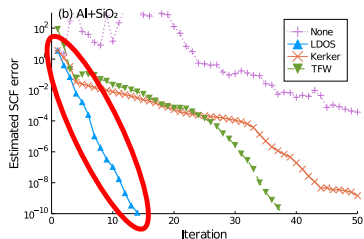
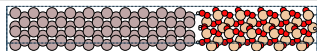
The screenshot shows two GitHub pull requests for the JuliaMolSim/DFTK.jl repository. The top PR, #711, is titled "Make LOBPCG GPU-compatible" and has 10 commits. The bottom PR, #712, is titled "Make some computations in DFTK GPU-compatible" and has 1 commit. Both PRs show a "155 -36" status, indicating 155 lines added and 36 lines removed. The #712 PR includes a comment from @Wgme dated 24 Aug, discussing the implementation of GPU compatibility for LOBPCG and the goal of making the code work on GPUs. A code snippet in the comment shows a Julia function call: `basis = PlaneWaveBasis(model; Ecut=30, kgrid=(1, 1, 1), architecture=DFTK.GPU(CuArray))`. The PR #712 also shows a commit message: "155 -36" and a commit hash: "155 -36".

- Use **julia**'s HPC abstractions to target all of CUDA, ROCm, oneAPI
- < 500 lines changed
- Collaboration with **julia** lab: CS, physics & maths
- 10-week GSoC project

```
basis = PlaneWaveBasis(model; Ecut=30, kgrid=(1, 1, 1),  
                        architecture=DFTK.GPU(CuArray))
```

- Note: **julia** allows seamless composition of
 - Floating-point agnostic code for computing arbitrary derivatives (algorithmic differentiation), guaranteed error control (intervals), etc.
 - Fast code integrating with MPI, CUDA, ...

Robust & efficient algorithms



- Preconditioning inhomogeneous systems (surfaces, clusters, ...)
- LDOS preconditioner¹:
Parameter-free and self-adapting
- ca. 50% less iterations



- First-principle properties of metals
- Schur-complement approach to perturbation theory²
(exploits partially converged states)
- ca. 40% less iterations

⇒ Maths / physics collaboration:

Exchange of ideas between simplified & practical settings crucial

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

²E. Cancès, MFH, G. Kemlin, et. al. Lett. Math. Phys. **113**, 21 (2023).

-  **DFTK**: Mathematical efforts on **DFT** modelling:
 - Self-adapting black-box DFT methods^{1,2}
 - Numerical analysis of DFT^{3,4}
 - Practical error bounds^{5,6}
- github.com/ACEsuit: **Atomic Cluster Expansion**⁷
 - Collaboration mathematics & applications
- github.com/JuliaMolSim/Molly.jl: **Molecular dynamics**
 - Collaboration CS & application
- **Cross-disciplinary community** efforts: JuliaMolSim & AtomsBase.jl
 -  interfaces and data structures for materials modelling
- Overview talk: **Julia for Materials Modelling** (youtube recording)
 - <https://github.com/mfherbst/julia-for-materials>

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

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

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




⁴E. Cancès, MFH, G. Kemplin, *et. al.* Lett. Math. Phys. **113**, 21 (2023).

⁵MFH, A. Levitt, E. Cancès. Faraday Discuss. **223**, 227 (2020).

⁶E. Cancès, G. Dusson, G. Kemplin *et. al.* SIAM J. Sci. Comp., **44**, B1312 (2022).

⁷R. Drautz. Phys. Rev. B **99**, 014104 (2019).

Summary

- Challenges in materials modelling
 - Inherently interdisciplinary research regime (e.g. high-throughput)
 - Codes frequently focus on single community
 - **Integration & collaboration barrier**
- **People compose if software composes**
 - Cross-disciplinary ideas should not fail due to software
 - Key ingredient: Separating **what** and **how**
 - ⇒ Better collaboration by separation of concern
- What makes  codes so composable?
 - **Specialisation**: Performance & hardware specifics
 - **Abstraction**: Code becomes the math
 - **Multiple dispatch**: Repurpose existing code (e.g. AD)
- Experience with -based materials codes:
 - Concise, accessible & HPC ready
 -  **DFTK**: One code for reduced problems & applications

Acknowledgements

- Antoine Levitt (Université Paris-Saclay)
- Alan Edelman (MIT)
- Valentin Churavy (MIT)
- All  DFTK contributors




Open PhD & PostDoc positions in the MatMat group



Possible topics include:

- **Uncertainty quantification for DFT:**
Error in data-driven DFT models, pseudopotentials, propagation to properties and MD potentials
- **Self-adapting numerical methods** for high-throughput DFT simulations
- See <https://matmat.org/jobs/>


- **Interdisciplinary research** linking maths and simulation:
 - Become part of maths **and** materials institutes @ EPFL


- Collaboration inside  MARVEL


- Reproducible workflows & sustainable software
- Computational materials discovery
- Statistical learning methods





Questions?


 <https://matmat.org>

 mfherbst

 michael.herbst@epfl.ch

 <https://michael-herbst.com/slides/pasc23>

 **DFTK** <https://dftk.org>

 **Julia** <https://github.com/mfherbst/julia-for-materials>
<https://michael-herbst.com/learn-julia>