

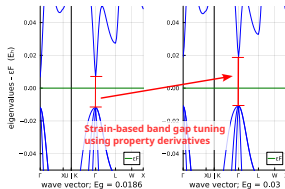
The Julia language in atomistic modelling: Software integration across research communities

Michael F. Herbst

Mathematics for Materials Modelling (matmat.org), EPFL

20 January 2026

https://michael-herbst.com/talks/2026.01.20_CECAM_ML_Roadmap.pdf



Acknowledgements

 XtMat group

- Bruno Ploumhans
- Nathanael Bosch
- Niklas Schmitz



AtomsBase/AtomsCalculators

- Christoph Ortner (UBC)
- Rachel Kurchin (CMU)
- Teemu Järvinen (UBC)
- Joe Greener (Cambridge)
- Spencer Wyant (MIT)
-  JuliaMolSim community



Magic of : 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*
- **Note**: We can even use “**new Hows**” ... such as AD

New Hows: Code reinterpretation & self-implementing features

```
using OrdinaryDiffEq, Plots

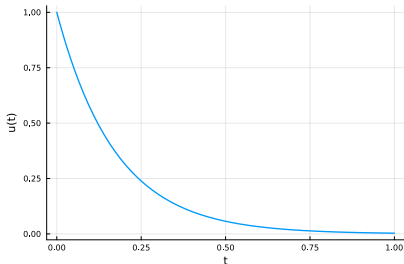
# Half-life of Carbon-14 is 5730 years.
c = 5.730

# 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)
```



New Hows: Code reinterpretation & self-implementing features

```
using OrdinaryDiffEq, Measurements, Plots

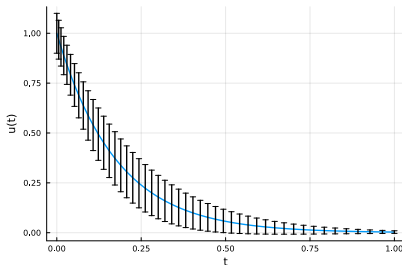
# Half-life of Carbon-14 is 5730 years.
c = 5.730 ± 2

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

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

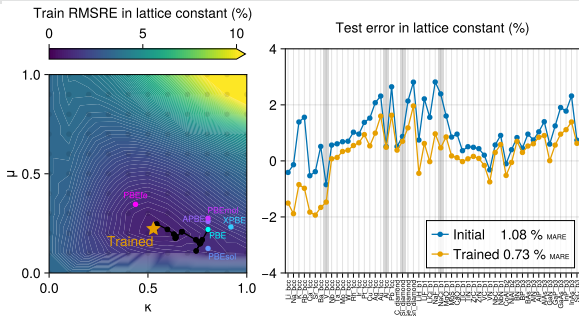
# Pass to solver
prob = ODEProblem(radioactivedecay, u0, tspan)
sol = solve(prob, Tsit5();
            reltol=1e-8, abstol=1e-8)

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”

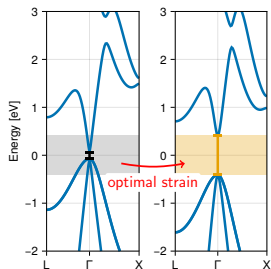
New hows: DFT inverse design using AD-DFPT¹



Inverse design of PBE-type model wrt. experimental lattice constant¹

- Tune **band gap** towards target
- Derivative:
Band gap versus **strain**
(lattice distortion)

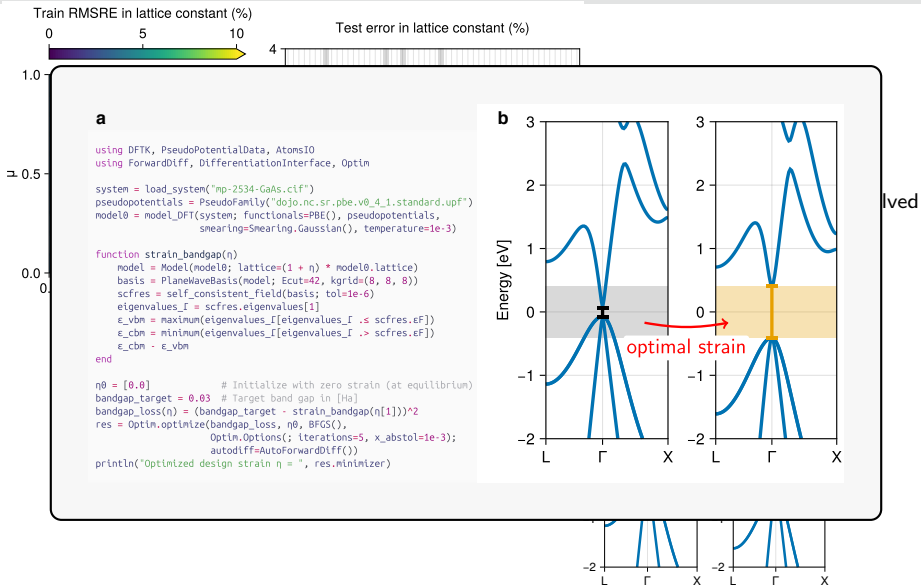
- Relaxed geometries
- SCF & geometry optimisation loop involved



Inverse design structure wrt. desired property¹

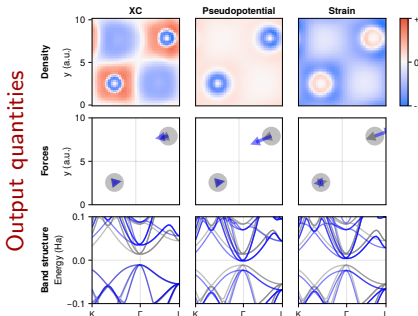
¹N. Schmitz, B. Ploumhans, MFH. npj Computat. Mater. 12, 6 (2025).


New hows: DFT inverse design using AD-DFPT¹

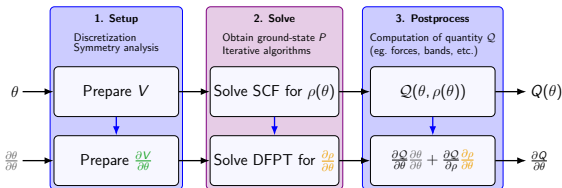


Inverse design structure wrt. desired property¹

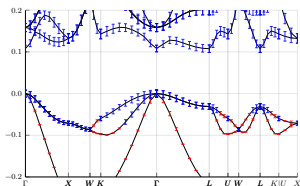
Input parameters θ



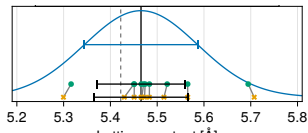
Sensitivity of key DFT output quantities computed using AD in  DFTK



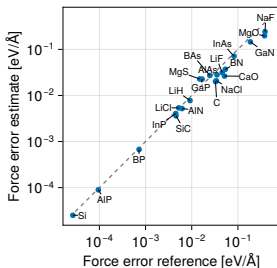
Larger thrust: Quantifying DFT simulation error



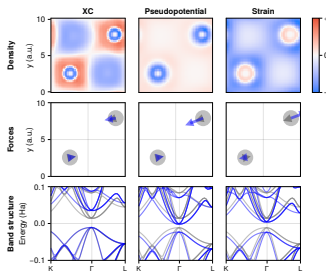
Band structure with **guaranteed error bars**¹



DFT lattice constant **error distribution**²



Plane-wave **basis error estimates** at 20Ha^{2,3}



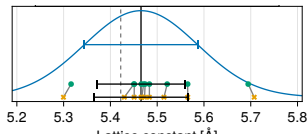
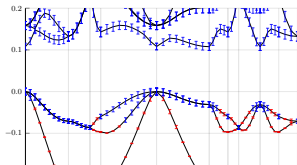
DFT quantity vs. **parameter sensitivities**²

¹MFH, A. Levitt, E. Cancès. Faraday Discuss. **223**, 227 (2020).

²N. Schmitz, B. Ploumhans, MFH. npj Computat. Mater. **12**, 6 (2025).

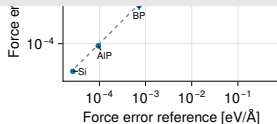
³E. Cancès, G. Dusson, G. Kemplin et. al. SIAM J. Sci. Comp., **44**, B1312 (2022).

Larger thrust: Quantifying DFT simulation error

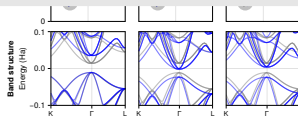


Towards **inexact computation without harm**:

- Error remains controllable
- Challenge: Turning **mathematical analysis into practical tool**
 - Needs performance tuning, heuristics, best practices, ...
- **Re-purposing julia code (new hows)** is a crucial component



Plane-wave **basis error estimates** at $20\text{Ha}^{2,3}$



DFT quantity vs. **parameter sensitivities**²

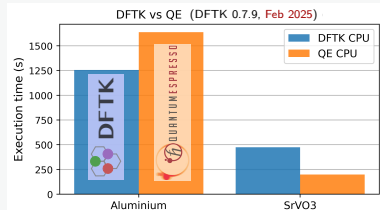
¹MFH, A. Levitt, E. Cancès. Faraday Discuss. **223**, 227 (2020).

²N. Schmitz, B. Ploumhans, MFH. npj Computat. Mater. **12**, 6 (2025).

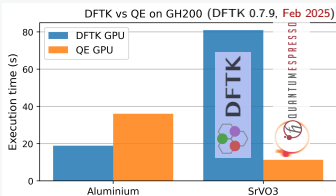
³E. Cancès, G. Dusson, G. Kemplin et. al. SIAM J. Sci. Comp., **44**, B1312 (2022).



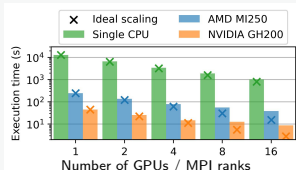
CPU performance



Nvidia performance






- Ported to **Nvidia & AMD GPUs**
 - **Performance without bloat:**
Net added just ≈ 400 lines
 - **Julia** HPC ecosystem
- GPU speedup up to **250x** (on GH200)
 - **Incremental perf improvements**
 - Similar or faster than QE
 - **Multi-GPU** support
- Only **10 000 lines** of code
 - Accessible to math. research



(Note: Workload does not saturate 16 GPUs)





Augustin Bussy
(CSCS)

- Magic of  julia: Separation of **what** and **how**:
 - Enables unusual code *reinterpretation*
(Algorithmic differentiation, symbolics, cross-platform compilation)
 - Separation of concern
Algorithms (mathematics), Models (physics), Scripting (application)
 - Cross-disciplinary **expertise can compose** in one code
 - **Hypothesis: People compose if software composes**
- Modelling and algorithm code stays high-level
 - Appropriate **specialisations unlock performance**
 - We can **add them gradually** as needed (Iterative optimisation)
-  JuliaMolSim: **Keep composability** in interface
-  JuliaMolSim: **Integration** across ecosystems:
 - Vast portfolio of relevant methods (e.g. in LAMMPS, ASE, QE)

⇒ Need two-way communication (for testing & adoption!)


DEMO

Using  **JuliaMolSim** and  **DFTK** in practice



→ https://michael-herbst.com/talks/2026.01.20_CECAM_ML_Roadmap_demo.tar.gz

julia materials ecosystem: What you can do today

- Interact with **AbstractSystems**:
 - **AtomsBuilder.jl**: Build structures (surfaces, defects, ...)
 - **AtomsIO.jl**: Read/write structure files
 - **ASEconvert.jl**: Use **ASE** Atoms
- Some **AtomsCalculators**:
 - **DFTK.jl**: Density-functional theory (1D & 1000 electrons, MPI, GPU)
 - **EmpiricalPotentials.jl**: LJ, StillingerWeber, Morse, ...
 - **acesuit**: ACE-based machine learning potentials
 - **IPICalculator.jl**: Use calculators via  **interface**
 - **ASEconvert.jl**: Use **ASE** calculators
- Dynamics and post-processing:
 - **GeometryOptimization.jl**: Opt. structures using **generic optimisers**
 - **Molly.jl**: Differentiable, GPU-enabled molecular dynamics
 - **Wannier.jl**: Localisation of electronic structures
- Export to external ecosystems:
 - E.g. **IPICalculator.jl**, **LAMMPS.jl**
 - **julitorch** (Integration with PyTorch), **Reactant.jl** (Julia to JAX)

Synthesis

- **Composability** is key
 - Which code is **high-level** and **low-level** depends on community (Mathematicians work on algorithms, physicists on models)
 - Focus on interfaces, rather than enforcing *one* implementation (Error estimates and UQ / caching, what properties matter ?)
 - Don't reinvent the wheel: **Cross-ecosystem integration**
- What **julia** can bring to the table
 - Avoid **two-language problem**
 - Clear Separation of what & how
- DFT error estimation capabilities
 - **Exploit error** during MLIP training
 - Error **propagation** and sensitivities **all the way**
- Software design and community building is **real work**:
 - Coordination of efforts **frustrating** and **time consuming**
 - Who writes **documentation** ?

Questions?



https://michael-herbst.com/talks/2026.01.20_CECAM_ML_Roadmap.pdf



<https://matmat.org>



mfherbst



@herbst @social.epfl.ch












michael.herbst@epfl.ch



JuliaMolSim

- <https://juliamolsim.org>
- <https://juliamolsim.zulipchat.com/register/>
- Monthly calls ...

Abstracting the how: HPC libraries

Accelerators	Shared Mem	Distributed
 CUDA.jl	 AMDGPU.jl	 OneAPI.jl
 JuliaGPU/ GPUArrays.jl Reusable array functionality for Julia's various GPU backends.		 JuliaFolds/ FLoops.jl Fast sequential, threaded, and distributed for-loops for Julia-fold for humans™ Announcing composable multi-threaded parallelism in Julia 23 July 2018 Jeff Bezanson (Julia Computing), Aronnek Nook (Julia Computing), Kien Nguyen (Intel) Base / Multi-Threading Multi-Threading  Base.Threads.@threads – Macro
https://github.com • JuliaGPU • KernelAbstractions KernelAbstractions.jl - Heterogeneous programming in Julia Heterogeneous programming in Julia. Contribute to JuliaGPU/KernelAbstractions.jl development by creating an account on GitHub.		 GitHub - eth-cscs/mpi4j: MPI wrappers for Julia  JuliaParallel/ Dagger.jl A framework for out-of-core and parallel execution Standard Library / Distributed Computing
 Julia Atomics Manifesto This proposal aims to define the memory model of Julia and to provide certain guarantees to the consumers of data races, both by default and through providing hooks to allow the user to specify the level of guarantees required. This should allow native implementations on Julia to employ system primitives (like mutexes), integrate with native hardware capabilities, and also to give generally reusable behaviors without incurring significant performance cost. Additionally, it should be the general audience and not clear about the user to identify opportunities with respect to ensuring that an atomic-type field is accessed with proper care for synchronization.		Distributed Computing

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

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

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

```

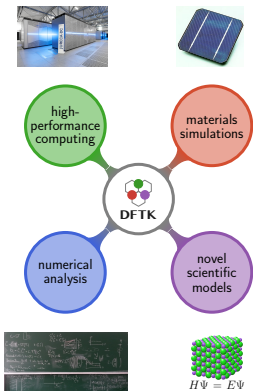
```
using CUDA
power_method(itinv(CuArray(A)), CuArray(x))

```

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

```

Density-functional toolkit¹ — <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 **10 000 lines** of code¹
 - Close the gap: **Maths** ↔ **high-throughput**:
AiiDA plugin
- **Fully composable** due to **Julia** abstractions:
 - Algorithmic differentiation (AD)
 - HPC tools: MPI, **Nvidia & AMD** GPUs
- **High-productivity** framework & established **community**:
 - Over 50 contributors in 6 years (Maths, physics, CS, ...)
 - Instrumental in a dozen of research works
- **Unique features**¹:
 - Self-adapting algorithms
 - Algorithmic differentiation
 - Numerical error estimates (e.g. basis set error in forces)

¹<https://dftk.org/features>