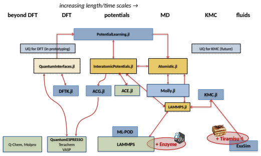
Fostering interdisciplinary research by composable julia software

Michael F. Herbst

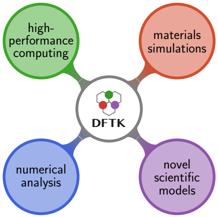
Mathematics for Materials Modelling (`matmat.org`), EPFL

27 June 2023

Slides: `https://michael-herbst.com/slides/pasc23`



Real-world multi-physics
software stack for materials modelling



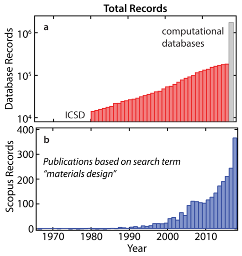
$$\text{Stress} = \frac{1}{\det(\mathbf{L})} \left. \frac{\partial E\left[P_*, (I + \mathbf{M})\,\mathbf{L}\right]}{\partial \mathbf{M}} \right|_{\mathbf{M}=0}$$

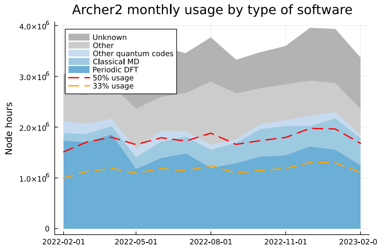
```
# Run SCF, get P*
scfres = self_consistent_field(basis)
     L = basis.model.lattice
stress = 1/det(L) * gradient(
    M -> recompute_energy(
                scfres, (I + M) * L),
    zero(L)
)
```

julia vision: Math ≡ 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





K. Alberi *et. al.* J. Phys. D, **52**, 013001 (2019).

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

FireWorks AFLOW Automatic-FLOW for Materials Discovery MATERIALSCLOUD AiiDA NOMAD NOVEL MATERIALS DISCOVERY pyiron

Minisymposium MS3D @ PASC23

Interdisciplinary Challenges in Multiscale Materials Modeling

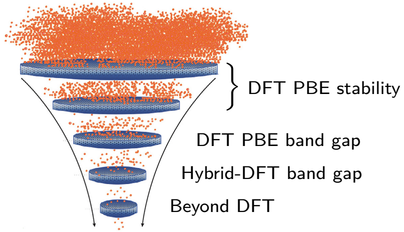
. . . and the role of software in overcoming them

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

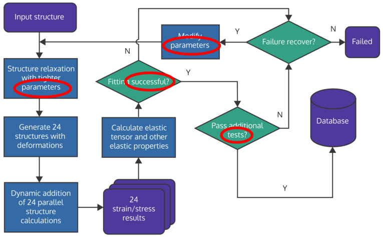
Contents

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

Sketch of high-throughput workflows



Design funnel for photovoltaic materials


Input structure
Structure relaxation with tighter parameters
Generate 24 structures with deformations
Dynamic addition of 24 parallel structure calculations
24 strain/stress results
Calculate elastic tensor and other elastic properties
Fitting successful?
Modify parameters
Failure recover?
Failed
Pass additional tests?
Database
Y
N
Workflow for computing elasticity tensors

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

G. Hautier Comput. Mater. Sci. **164**, 108 (2019); L. Himanen *et. al.* Adv. Science **6**, 1900808 (2019).

(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

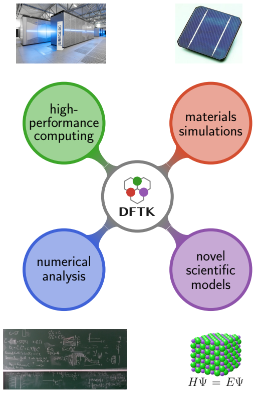
- **Goal:** Modelling physics
- **Scope:** All relevant systems
- Mix of **languages:** C, FORTRAN, python, . . .
- **Lifetime:** 100 manyears
- **Size:** 100k – 1M lines
- Obliged to write performant code

- 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 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
  - $\Rightarrow$ 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

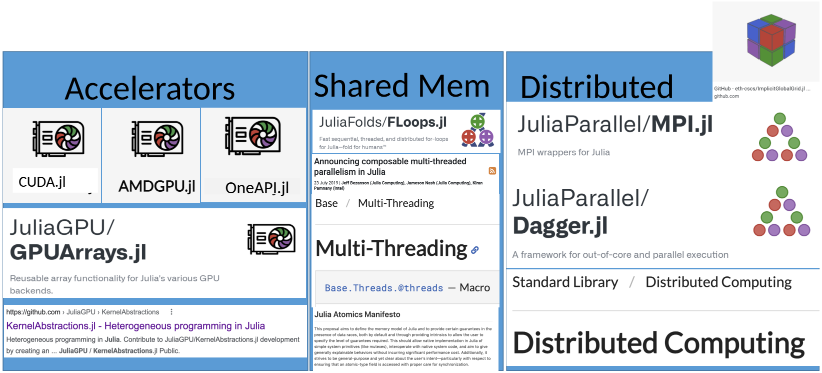
- Why is this separation so important ...
  - ... for composable software?
  - ... for multidisciplinary research?

- 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

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

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

JuliaParallel/**MPI.jl**

MPI wrappers for Julia

JuliaParallel/**Dagger.jl**

A framework for out-of-core and parallel execution

Standard Library / Distributed Computing

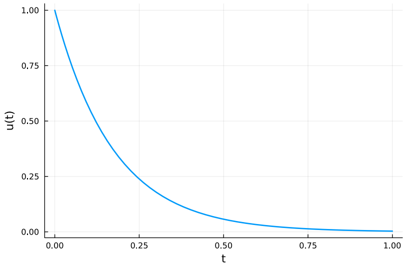
Distributed Computing

```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))
```

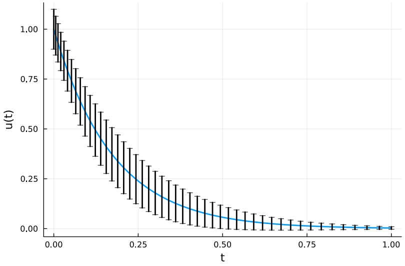
Code reinterpretation & self-implementing features

```
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"

julia and composable software

- 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

Contents

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

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

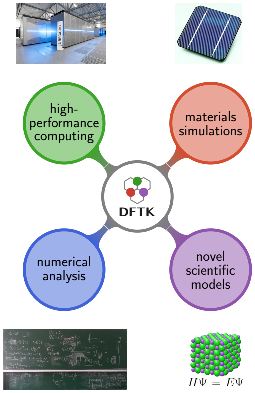
nuclear attraction V_{nuc}, exchange-correlation V_{XC}, Hartree potential $-\Delta\left(v_C \rho\right) = 4\pi\rho$

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

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

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

Density-functional toolkit[1] — `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 undegrads / PhD students
  - AD support in 10 weeks (CS Bachelor)
  - GPU support in 10 weeks (Physics Bachelor)
  - Relevant contributions from outside collab. circle

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)
)
```

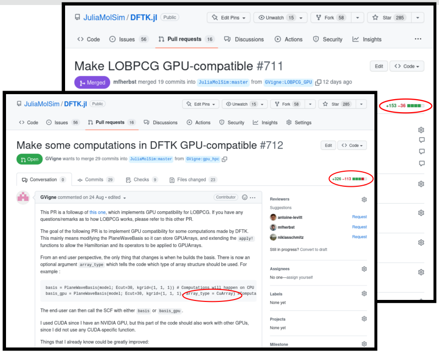
- Stress computation (Definition *vs.* julia code)[1]
- Post-processing step ⇒ Not performance critical
- Comparison of implementation complexity:
    - DFTK : 20 lines[1] (forward-mode algorithmic differentiation)
    - Quantum-Espresso: 1700 lines[2]
    - ≃ 10-week GSoC project

⇒ 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

GPU support in DFTK



Make LOBPCG GPU-compatible #711

Merged mfherbst merged 19 commits into `JuliaMolSim:master` from `GVigne:LOBPCG_GPU` 12 days ago

Make some computations in DFTK GPU-compatible #712

Open GVigne wants to merge 29 commits into `JuliaMolSim:master` from `GVigne:gpu_hpc`

GVigne commented on 24 Aug • edited

This PR is a followup of this one, which implements GPU compatibility for LOBPCG. If you have any questions/remarks as to how LOBPCG works, please refer to this other PR.

The goal of the following PR is to implement GPU compatibility for some computations made by DFTK. This mainly means modifying the PlaneWaveBasis so it can store GPUArrays, and extending the `apply!` functions to allow the Hamiltonian and its operators to be applied to GPUArrays.

From an end user perspective, the only thing that changes is when he builds the basis. There is now an optional argument `array_type` which tells the code which type of array structure should be used. For example :

```
basis = PlaneWaveBasis(model; Ecut=30, kgrid=(1, 1, 1)) # Computations will happen on CPU
basis_gpu = PlaneWaveBasis(model; Ecut=30, kgrid=(1, 1, 1), array_type = CuArray) #Computa
```

The end-user can then call the SCF with either `basis` or `basis_gpu`.

I used CUDA since I have an NVIDIA GPU, but this part of the code should also work with other GPUs, since I did not use any CUDA-specific function.

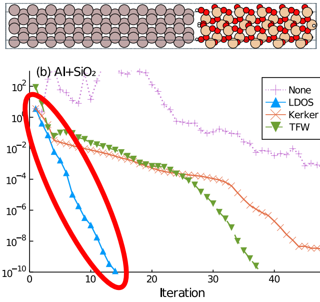
Things that I already know could be greatly improved:

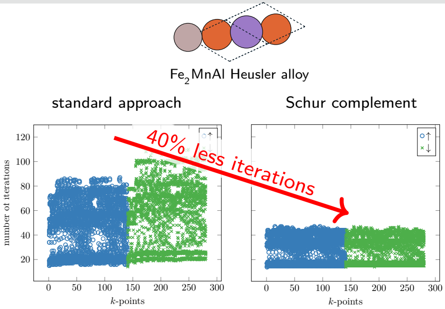
- Use julia's HPC abstractions to target all of CUDA, ROCm, oneAPI
- $<$ 500 lines changed
- Collaboration with julia lab: CS, physics & maths
- 10-week GSoC project

```
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 preconditioner[1]: Parameter-free and self-adapting
- ca. 50% less iterations

- First-principle properties of metals
- Schur-complement approach to perturbation theory[2] (exploits partially converged states)
- 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]E. Cancès, MFH, G. Kemlin, *et. al.* Lett. Math. Phys. **113**, 21 (2023).

julia materials codes: Bringing communities together

- **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`
  - julia interfaces and data structures for materials modelling
- Overview talk: Julia for Materials Modelling (youtube recording)
  - `https://github.com/mfherbst/julia-for-materials`

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

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

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

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

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

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

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

- Experience with julia-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

EPFL MatMat Summer of code

Applied and Computational Mathematics | RWTH AACHEN UNIVERSITY MIT CESMIX

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

https://github.com/mfherbst/julia-for-materials

https://michael-herbst.com/learn-julia