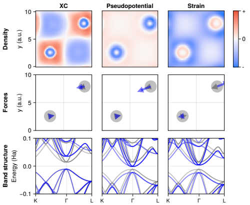
Algorithmic differentiation in plane-wave DFT

Michael F. Herbst

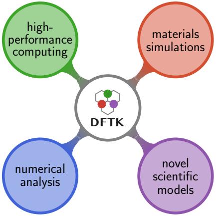
Mathematics for Materials Modelling (`matmat.org`), EPFL

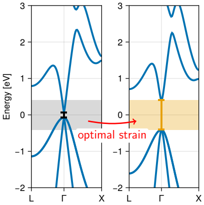
09 October 2025

`https://michael-herbst.com/talks/2025.10.09_IPAM_DFT_Gradients.pdf`



Gradients of DFT quantities
wrt. DFT parameters





Gradient-driven inverse design

Acknowledgements

MXtMat group

Niklas Schmitz

Bruno Ploumhans

Response algorithms

- Eric Cancès (École des Ponts)
- Gaspard Kemlin (Université de Picardie)
- Antoine Levitt (Université Paris-Saclay)
- Benjamin Stamm (Stuttgart)
- Bonan Sun (EPFL & MPI Magdeburg)

Fonds national suisse

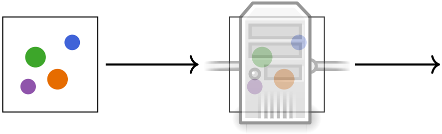
MARVEL

NATIONAL CENTRE OF COMPETENCE IN RESEARCH

EPFL MXtMat

Ab initio materials modelling

$$\min_{\Psi} \langle \Psi, H\Psi \rangle$$

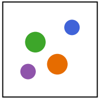


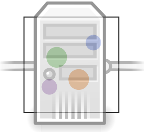
Suitable for solar cells?

Ab initio materials modelling

$$\min_{\Psi} \langle \Psi, H\Psi \rangle$$

Properties





Suitable for solar cells?

Density-functional theory: The textbook picture

- DFT approximation: Effective single-particle model

$$\begin{cases} \forall i \in 1 \dots N: \left(-\frac{1}{2}\Delta + V\left(\rho_\Phi\right)\right)\psi_i = \varepsilon_i \psi_i, \\ \qquad V(\rho) = V_{\text{ext}} + V_{\text{Hxc}}(\rho), \quad \text{where } V_{\text{Hxc}}(\rho) = v_C\,\rho + V_{\text{XC}}(\rho) \\ \qquad \rho_\Phi = \sum_{i=1}^{N} f\left(\frac{\varepsilon_i - \varepsilon_F}{T}\right)|\psi_i|^2, \end{cases}$$

- Self-consistent field procedure: Fixed-point problem

$$F\big(V_{\text{ext}} + V_{\text{Hxc}}(\rho_{\text{SCF}})\big) = \rho_{\text{SCF}}$$

- $F(V)$ is the potential-to-density map (i.e. diagonalisation)

$$F(V) = \sum_{i=1}^{\infty} f\left(\frac{\varepsilon_i - \varepsilon_F}{T}\right)|\psi_i|^2 \quad \text{where} \quad \left(-\frac{1}{2}\Delta + V\right)\psi_i = \varepsilon_i \psi_i$$

- ε_F chosen such that $\int F(V) = N$ (number of electrons)

- nuclear attraction V_{nuc}, exchange-correlation V_{XC}, Hartree potential $-\Delta\left(v_C\,\rho\right) = 4\pi\rho$, ψ_i orthogonal, f: Occupation function between 0 and 2

Materials properties: Simulation $\leftrightarrow$ experiment

- DFT properties: Response of system to external changes:
  - Connection Theory $\Leftrightarrow$ Experiment
  - Modelling: Potential $V(\theta, \rho)$ depends on parameters θ (e.g. atomic positions, el. field)

- SCF procedure yields fixed-point density ρ_{SCF}

$$0 = F\Big(V(\theta, \rho_{\text{SCF}})\Big) - \rho_{\text{SCF}}$$

$\Rightarrow$ Defines implicit function $\rho_{\text{SCF}}(\theta)$

- Properties are derivatives:
  - **Forces** (energy wrt. position), **dipole moment** (energy wrt. el. field), **elasticity** (energy cross-response to lattice deformation), phonons, electronic **spectra**, ...

- But what about the following derivatives? (details in this talk)
  - θ is parameter of DFT model: ... **uncertainty quantification**
  - θ is design parameter (e.g. strain): ... **inverse materials design**
  - θ is parameter of discretisation: ... ***a posteriori* error estimates**

What about inverse design ?

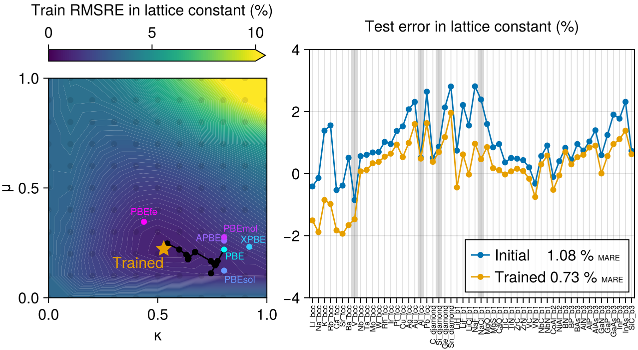
- θ: Design parameters (e.g. strain, dopant concentration)
- SCF procedure yields fixed-point density ρ_{SCF}
$$0 = \rho\Big(V(\theta, \rho_{\text{SCF}})\Big) - \rho_{\text{SCF}}$$
- From these compute quantities of interest: $Q(\rho)$

- **Question:** How to chose θ to reach target quantity ?
- **Answer:** Follow gradient

$$\frac{dQ}{d\theta} = \frac{\partial Q}{\partial \theta} + \frac{\partial Q}{\partial \rho}\frac{\partial \rho}{\partial \theta}$$

$\Rightarrow$ Need derivative of implicit function $\rho(\theta)$

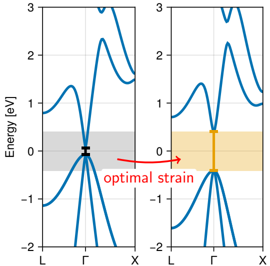
Inverse design showcases using AD-DFPT[1]



Inverse design of PBE-type model wrt. experimental lattice constant[1]

- Relaxed geometries
- SCF & geometry optimisation loop involved

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



[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

What about DFT uncertainty propagation ?

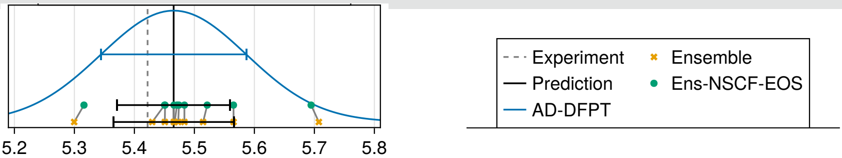
- θ: DFT model parameters (e.g. XC, pseudopotential)
- SCF procedure yields fixed-point density ρ_{SCF}
$$0 = \rho\Big(V(\theta, \rho_{\text{SCF}})\Big) - \rho_{\text{SCF}}$$
- From these compute quantities of interest: $Q(\rho)$

- **Question:** Knowing a distribution of θ, how to get distribution of Q ?
- Approx. **answer:** Linear propagation, e.g. if $\theta \sim N(0, \Sigma)$:

$$Q \sim N\left[Q(\rho_{\text{SCF}}), J\Sigma J^T\right] \quad \text{with} \quad J = \frac{dQ}{d\theta} = \frac{\partial Q}{\partial \theta} + \frac{\partial Q}{\partial \rho}\frac{\partial \rho}{\partial \theta}$$

$\Rightarrow$ Need again derivative of implicit function $\rho(\theta)$

What about DFT uncertainty propagation ?



Error propagation DFT model (BEEF) $\to$ geometry[1]

- BEEF model provides (posterior) distribution for θ
- Propagation to optimal lattice constant:

$$a^*(\theta) = \arg\min_a \left(\min_\rho \ \mathcal{E}(\theta, a, \rho) \right)$$

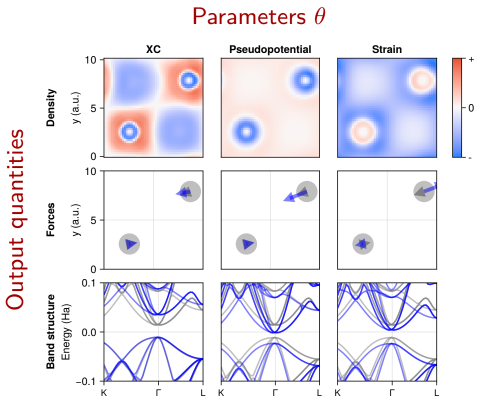
- Comparison of approaches:
  - Blue: DFPT-AD and linearisation[1]
  - Orange: MC / Ensemble: 10 times expensive optimisation
  - Green: Equation of state fit (approach specific to this property)

$\Rightarrow$ Ideally arbitrary Q should be possible, UQ should be fast

$\Rightarrow$ Consistent UQ across DFT quantities

[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

Combinatorial explosion problem



Sensitivity of key DFT output quantities computed using AD in DFTK[1]

- We should support all pairs Q vs. θ
- Core problem is computing $\rho'(\theta)$

[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

Ingredient 1: Density-functional perturbation theory

$$F(V_{\text{ext}} + V_{\text{Hxc}}(\rho_{\text{SCF}})) = \rho_{\text{SCF}}$$

- δV: Perturbation to V_{ext}, by chain rule

$$\begin{aligned} &\delta\rho = F'(V_{\text{ext}} + V_{\text{Hxc}}(\rho_{\text{SCF}})) \cdot (\delta V + K_* \delta\rho) \\ \Leftrightarrow \quad &\delta\rho = (1 - \chi_0 K)^{-1} \chi_0 \delta V \end{aligned}$$

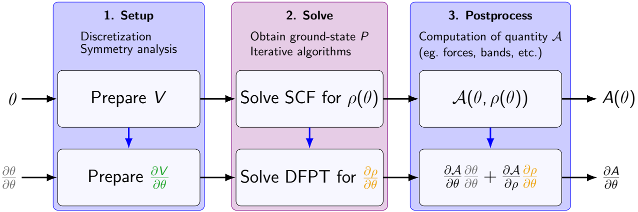
where $K_* = V'_{\text{Hxc}}(\rho_{\text{SCF}})$, $\chi_0 = F'(V_{\text{ext}} + V_{\text{Hxc}}(\rho_{\text{SCF}}))$

- Dyson equation: Solved by iterative methods

$\Rightarrow$ Density-functional perturbation theory (implicit differ$^{\text{n}}$.)

$$\frac{\partial \rho_{\text{SCF}}}{\partial \theta} = [1 - \chi_0 K]^{-1} \chi_0 \frac{\partial V}{\partial \theta}$$

Ingredient 2: Algorithmic differentiation (AD)

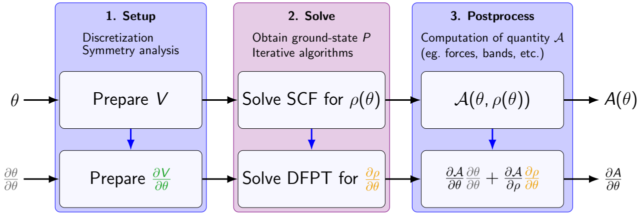


- Algorithmic differentiation (AD)[1]
  - Generic framework for derivatives: Request gradient, AD delivers
  - Chain rule: Automatically compose gradients out of primitives
    - *Primitives:* Floating-point operations, diagonalisation, SCF
  - Custom rule: For SCF problem gradient is DFPT

$$\frac{\partial\rho_{\text{SCF}}}{\partial\theta} = \left[1 - \chi_0 K\right]^{-1} \chi_0 \frac{\partial V}{\partial\theta}$$

[1]Remark: I only discuss **forward-mode** AD here, but I have bonus slides explaining **adjoint-mode** AD, which scales better for scalar outputs, ask if interested.

Tackling combinatorial explosion: AD+DFPT



$\Rightarrow$ Appropriate DFPT problem automatically set up and solved

- Each solver improvement available to all tasks !

- Separation of concern:
  - Mathematicians can work on smart algorithms for DFPT
  - Custom DFT gradients by simulation practitioners (DFT non-experts)

  $\Rightarrow$ Key ingredient to composable DFT codes

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

Algorithmic differentiation in DFT

- Multiple recent efforts to bring AD to practical DFT
  - DeepKS[1], DQC[2], pyscf[3], GradDFT[4], ...
  - Gaussian basis sets
  - Largely forks or disconnected code bases (Not all appear still maintained)
- Our approach: Density-functional toolkit (**DFTK** , `https://dftk.org`)
  - Plane-wave DFT (up to meta-GGA, UPF pseudos)[5]
  - 10 000 lines, GPU-enabled, differentiable DFT code
  - **Differentiability *not planned* from the start**

[1] Y. Chen, L. Zhang, H. Wang, W. E. J. Chem. Theo. Comp. **17**, 170 (2021)

[2] M. Kasim, S. Lehtola, S. Vinko. J. Chem. Phys. **156**, 084801 (2022).

[3] X. Zhang, G. Chan. J. Chem. Phys. **157**, 204801 (2022).

[4] M. Casares, J. Baker, M. Medvidović, *et. al.* J. Chem. Phys. **160**, 062501 (2024).

[5] `https://docs.dftk.org/features/`

DEMO

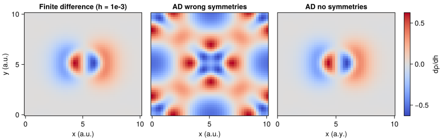
How AD works, AD-DFPT in action

→ https://michael-herbst.com/talks/2025.10.09_IPAM_DFT_Gradients_1_ad.html

→ https://michael-herbst.com/talks/2025.10.09_IPAM_DFT_Gradients_2_design.html

DFT gradients for everyone . . . but where's the catch ?

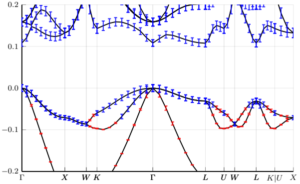
- **AD through DFT** is harder than AD through a neural net:
  - Requires robust and efficient DFPT solver for *arbitrary* RHS (not always so easy . . . next slides)
  - The gradient you ask for is **not** always the gradient you want.



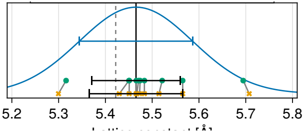
  - Taking gradients makes functions less smooth $\Rightarrow$ Numerical pathologies augment

- First-principle simulations: Non-differentiable intermediates
  - E.g. band structure crossings, Fermi level algos for insulators
  - But: Physical observables can still be differentiable
  - Even if not: Gradients can be useful (one sub/super-differential representative returned)

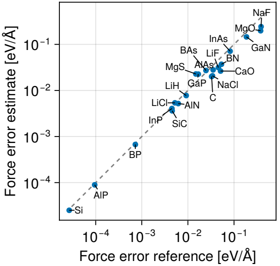
Larger thrust: Quantifying DFT simulation error



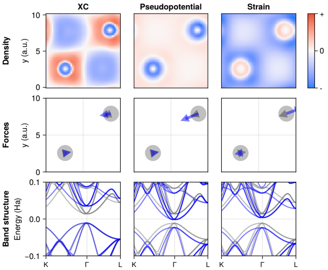
Band structure with guaranteed error bars[1]



DFT lattice constant error distribution[2]



Plane-wave basis error estimates at 20Ha[2,3]



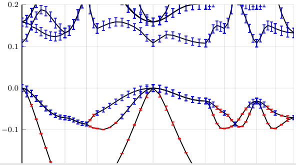
DFT quantity vs. parameter sensitivities[2]

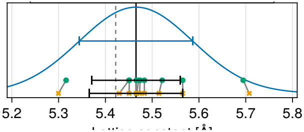
[1] MFH, A. Levitt, E. Cancès. Faraday Discus. **223**, 227 (2020).

[2] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

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

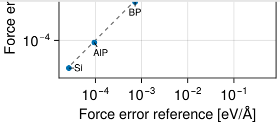
Larger thrust: Quantifying DFT simulation error



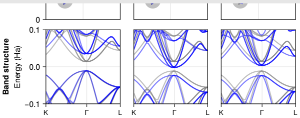


Towards **inexact computation without harm**:

- Error remains controllable
- Challenge: Turning mathematical analysis into practical tool
  - Needs performance tuning, heuristics, best practices, ...
- AD techniques are a crucial component



Plane-wave basis error estimates at 20Ha[2,3]



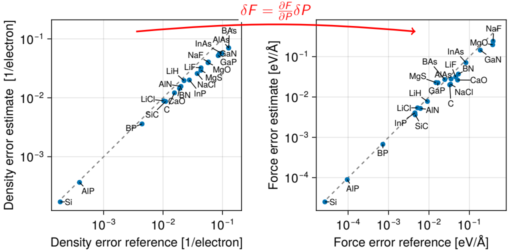
DFT quantity vs. parameter sensitivities[2]

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

[2] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

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

Illustration: Estimating the plane-wave basis error[1]



- Perturbation approach: Estimation of δP (Plane-wave error in density)[2]
  - Approximate first-order change of density P as Ecut increased
  - Cost of estimate scales only with size of *current* computational basis

- Propagation to QoIs: e.g. linear approximation $\delta F = \frac{\partial F}{\partial P}\delta P$

$\Rightarrow$ **DFTK** is major research tool[3,4]

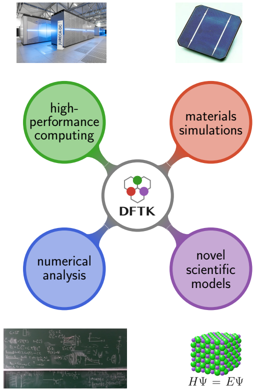
[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

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

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

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

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 10 000 lines of code[1]
  - 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:
  - 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://docs.dftk.org/features`

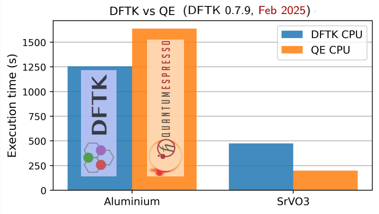
DEMO

DEMO

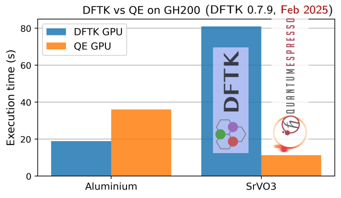
Custom models in DFTK

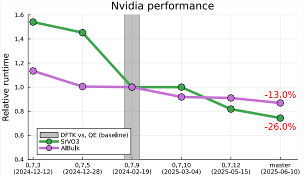
→ https://michael-herbst.com/talks/2025.10.09_IPAM_DFT_Gradients_3_dftk.html

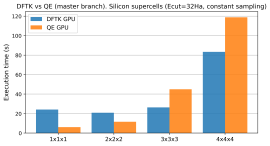
CPU performance



Nvidia performance



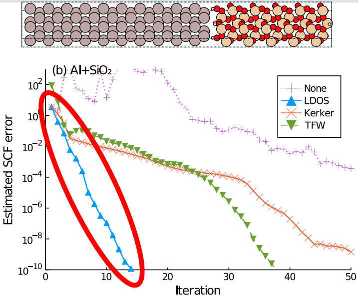


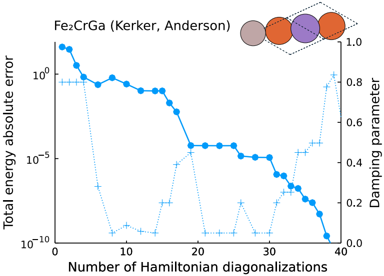


- Ported to Nvidia & AMD GPUs
  - **Performance without bloat**: Net added just $\simeq 400$ lines
  - julia HPC ecosystem
- DFTK sometimes beats QE
  - Speed within factor 2 throughout
  - **Incremental perf improvements**
- Only 10 000 lines of code
  - Accessible to math. research

Augustin Bussy (CSCS)

Self-adapting black-box algorithms





- Preconditioning inhomogeneous systems (surfaces, clusters, ...)
- LDOS preconditioner[1]: Parameter-free and self-adapting
- ca. 50% less iterations

- Damping α adapted *in each step* (using tailored quadratic model)
- Avoids trial and error (but may have a small overhead)
- Safeguard with theoretical guarantees[2]

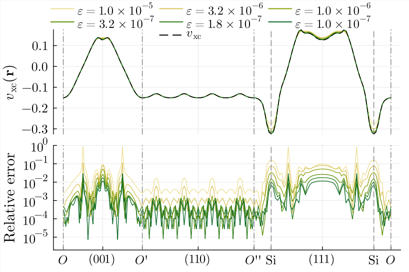
$\Rightarrow$ Maths / physics collaboration:
Exchange of ideas between simplified & practical settings crucial

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

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

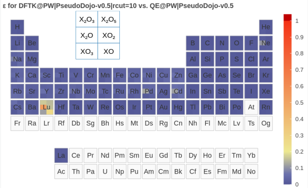
Other things you can do with DFTK

Kohn-Sham inversion[1]



- Find exact v_{xc} from exact ρ
- Based on Moreau-Yosida regularised formulation of DFT
- **Goal:** error bounds & mathematical guarantees[1]

Close gap: Math $\leftrightarrow$ high-throuhput



- DFTK plugin for AiiDA workflow manager
- Above: Verification of DFTK versus QE
- **Goal:** Simplify automated testing of novel algorithms

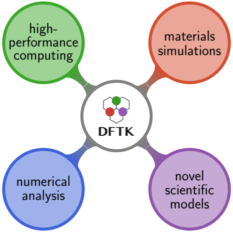
[1] MFH, V. Bakkestuen, A. Laestadius. Phys. Rev. B **111**, 205143 (2025).

Advertisement break: CECAM workshop

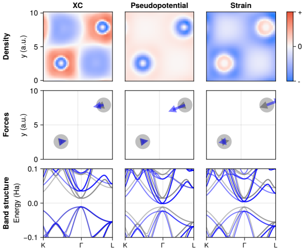
Topic: Uncertainty quantification in atomistic modelling

- **25 till 28 Nov 2025**
- DFT, ML surrogates & MD quantities of interest
- Quantification & propagation of errors across the scales
- See `https://www.cecam.org/workshop-details/1380`

Summary: AD in plane-wave DFT & DFTK



- **AD:** Unified framework for DFT gradients
  - Inverse model & material design[1]
  - DFT error estimation & propagation[1]



- **DFTK**: An AD-able plane-wave DFT code
  - Side effect of julia: Not a design choice
  - 10 000 lines, hackable & easy to extend
  - HPC support: MPI, Nvidia & AMD GPUs
  - Designed for cross-disciplinary research

EPFL MatMat

[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

Questions?

https://matmat.org

mfherbst

@herbst @social.epfl.ch

michael.herbst@epfl.ch

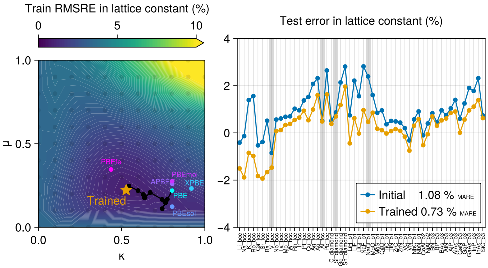
https://michael-herbst.com/talks/2025.10.09_IPAM_DFT_Gradients.pdf

N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

DFTK https://dftk.org

Contents

Details: XC optimisation



- Testing on Sol58LC dataset, Training on $C =$ Si, Al, V, NaCl
- Optimise PBE exchange (2 free parameters θ) to experimental lattice constants

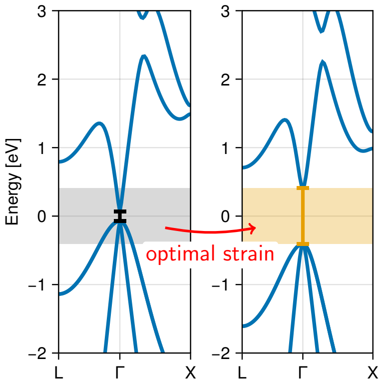
$$L_{\text{xc}}(\theta) = \frac{1}{4} \sum_{C} \left(\frac{a^*(C,\theta) - a_i^{\text{expt}}}{a_i^{\text{expt}}} \right)^2, \quad a^*(C,\theta) = \arg\min_a \left(\min_P \ \mathcal{E}(C,\theta,a,P) \right)$$

- Practical challenges for computation of $\frac{\partial L_{\text{xc}}}{\partial \theta}$:
  - Nested iterative methods (eigensolver, SCF, lattice optimisation)
  - Unusual second-order derivatives, e.g.

$$\frac{\partial a^*}{\partial \theta} = -\left(\frac{\partial^2 E}{\partial a^2} \right)^{-1} \left(\frac{\partial^2 E}{\partial \theta \partial a} \right)$$

Details: Band gap tuning

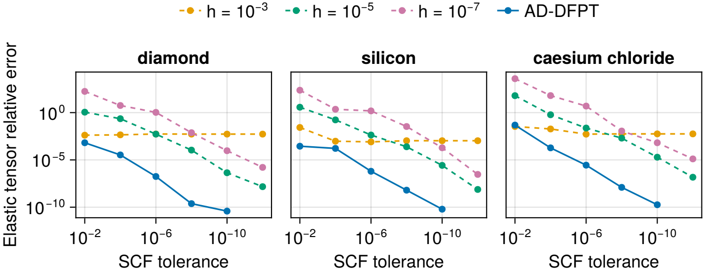
```julia
using DFTK, PseudoPotentialData, AtomsIO
using ForwardDiff, DifferentiationInterface, Optim

system = load_system("mp-2534-GaAs.cif")
pseudopotentials = PseudoFamily("dojo.nc.sr.pbe.v0_4_1.standard.upf")
model0 = model_DFT(system; functionals=PBE(), pseudopotentials,
                   smearing=Smearing.Gaussian(), temperature=1e-3)

function strain_bandgap(η)
    model = Model(model0; lattice=(1 + η) * model0.lattice)
    basis = PlaneWaveBasis(model; Ecut=42, kgrid=(8, 8, 8))
    scfres = self_consistent_field(basis; tol=1e-6)
    eigenvalues_Γ = scfres.eigenvalues[1]
    ε_vbm = maximum(eigenvalues_Γ[eigenvalues_Γ .≤ scfres.εF])
    ε_cbm = minimum(eigenvalues_Γ[eigenvalues_Γ .> scfres.εF])
    ε_cbm - ε_vbm
end

η0 = [0.0]              # Initialize with zero strain (at equilibrium)
bandgap_target = 0.03   # Target band gap in [Ha]
bandgap_loss(η) = (bandgap_target - strain_bandgap(η[1]))^2
res = Optim.optimize(bandgap_loss, η0, BFGS(),
                     Optim.Options(; iterations=5, x_abstol=1e-3);
                     autodiff=AutoForwardDiff())
println("Optimized design strain η = ", res.minimizer)
```



- This is the full implementation code
- Derivative computation implicitly requested by `Optim.optimize`

Contents

A word about accuracy: Elastic constants



- Accuracy superior to finite differences
- Implementation is literally[1] (Strain η)
  - Stress $\sigma(\eta) = \frac{1}{V(\eta)} \frac{\partial \mathcal{E}}{\partial \eta}$
  - Elastic constant $C(\eta) = \left.\frac{\partial \sigma}{\partial \eta}\right|_{\eta^*}$
- AD-DFPT provides ideal compromise between achieved accuracy and invested human time

[1] N. F. Schmitz, B. Ploumhans, MFH. *Algorithmic differentiation for plane-wave DFT*. arXiv:2509.07785

Contents

How does algorithmic differentiation (AD) work?

```
function F(x)
    y1 = x[1] + x[2]   # F1 = sum
    y2 = 2 * p         # F2 = double
    return y2
end
```

- Goal: Compute derivative of this code
- Function $F : \mathbb{R}^2 \to \mathbb{R}$ with $F(x) = \text{double}(\text{sum}(x_1, x_2))$
- Derivative at $\tilde{x}$ is characterised by its Jacobian matrix

$$[J_F(\tilde{x})]_{ij} = \left(\left. \frac{\partial F}{\partial x} \right|_{x=\tilde{x}} \right)_{ij} = \left. \frac{\partial F_i}{\partial x_j} \right|_{x=\tilde{x}}$$

- Finite differences: Simple, one column at a time:

$$[J_F(\tilde{x})]_{:,j} = \frac{F(\tilde{x} + \alpha e_j) - F(\tilde{x})}{\alpha}$$

(with e_i unit vectors)

$\Rightarrow$ Inaccurate and slow ($\mathcal{O}(N)$ times primal cost)

Chain rule to the rescue!

```
function F(x)
    y1 = x[1] + x[2]   # F1 = sum
    y2 = 2 * p         # F2 = double
    return y2
end
```

$$F(x) = \text{double}(\text{sum}(x_1, x_2))$$

- "double" and "sum" are simple and frequent primitives

$\Rightarrow$ Key idea of AD:

  - Compose the derivative of F from the Jacobians of primitives
  - Assumed to be known and already implemented

- Use chain rule as glue, e.g. for a Jacobian element at $\tilde{x}$:

$$\frac{\partial F_i}{\partial x_j} = \frac{\partial \text{double}(a)}{\partial a} \left(\frac{\partial \text{sum}(c,d)}{\partial c} \frac{\partial x_1}{\partial x_j} + \frac{\partial \text{sum}(c,d)}{\partial d} \frac{\partial x_2}{\partial x_j} \right)$$

- More compact: $e_i^T J_F e_j = e_i^T J_{\text{double}} J_{\text{sum}} e_j$
- Note: J_{double} is needed at $\text{sum}(\tilde{x}_1, \tilde{x}_2)$

Forward-mode algorithmic differentiation

```
function F(x)
    y1 = x[1] + x[2]  # F1 = sum
    y2 = 2 * p        # F2 = double
    return y2
end
```

$$F(x) = \text{double}(\text{sum}(x_1, x_2))$$

$$e_i^T J_F e_j = e_i^T J_{\text{double}} J_{\text{sum}} e_j$$

- Forward-diff: Evaluate in order with *primal* F:
  1. Set $y_0 = (x_1, x_2)$, $\dot{y}_0 = e_j$
  2. Compute $y_1 = \text{sum}(y_0)$ and $\dot{y}_1 = J_{\text{sum}}(y_0)\dot{y}_0$
  3. Compute $y_2 = \text{double}(y_1)$ and $\dot{y}_2 = J_{\text{double}}(y_1)\dot{y}_1$
  4. Obtain $F(x_1, x_2)$ as y_2 and $[J_F]_{:,j} = \dot{y}_2$

$\Rightarrow$ Again one column of J_F at a time

- Implementation: Numbers $\rightarrow$ dual numbers
- Vectorisation & other tricks: Usually faster than finite diff.
- But: Still $\mathcal{O}(N)$ times primal cost

Optimal cost for differentiation (1)

```
function F(x)
    y1 = x[1] + x[2]   # F1 = sum
    y2 = 2 * p         # F2 = double
    return y2
end
```

$$F(x) = \text{double}(\text{sum}(x_1, x_2))$$
$$e_i^T J_F e_j = e_i^T J_{\text{double}} J_{\text{sum}} e_j$$

Proposition

If $f : \mathbb{R}^N \to \mathbb{R}$ is a differentiable function, computing $\nabla f = J_f$ is asymptotically not more expensive than f itself.

$\Rightarrow$ This is violated for finite diff and forward diff.

- Let's try to be more clever:
  - We could write $F(x) = b^T A x$ for appropriate (sparse) A, b
  - Equivalent formulation: $F(x) = (A^T b)^T x$
  - Differentiate that: $\nabla F = A^T b$ $\Rightarrow$ costs the same as F.
- To generalise this idea note that (for scalar functions)

$$F(x) = b^T J_F x + O(x^2)$$

Optimal cost for differentiation (2)

```
function F(x)
    y1 = x[1] + x[2]  # F1 = sum
    y2 = 2 * p        # F2 = double
    return y2
end
```

$$F(x) = \text{double}(\text{sum}(x_1, x_2))$$
$$e_i^T J_F e_j = e_i^T J_{\text{double}} J_{\text{sum}} e_j$$

- Let's try to be more clever:
  - We could write $F(x) = b^T A x$ for appropriate (sparse) A, b
  - Equivalent formulation: $F(x) = (A^T b)^T x$
  - Differentiate that: $\nabla F = A^T b \;\Rightarrow\;$ costs the same as F.
- To generalise this idea note that (for scalar functions)

$$F(x) = b^T J_F x + \mathcal{O}(x^2) \qquad \text{with } b = e_1 = 1$$

$\Rightarrow$ Focus on computing adjoint of Jacobian:

$$e_i^T J_F e_j = \left(J_F^T e_i\right)^T e_j = \left(J_{\text{sum}}^T J_{\text{double}}^T e_i\right)^T e_j$$

Adjoint-mode algorithmic differentiation

```
function F(x)
    y1 = x[1] + x[2]  # F1 = sum
    y2 = 2 * p        # F2 = double
    return y2
end
```

$$F(x) = \text{double}(\text{sum}(x_1, x_2))$$

$$e_i^T J_F e_j = \left(J_{\text{sum}}^T J_{\text{double}}^T e_i\right)^T e_j$$

- Adjoint-mode AD: Derivative in reverse instruction order.
- *Forward pass*:
  1. Set $y_0 = (x_1, x_2)$
  2. Compute $y_1 = \text{sum}(y_0)$ and store it
  3. Compute $y_2 = \text{double}(y_1)$ and store it
- *Reverse pass*:
  1. Set $\bar{y}_2 = e_i$
  2. Compute $\bar{y}_1 = [J_{\text{double}}(y_1)]^T \bar{y}_2$
  3. Compute $\bar{y}_0 = [J_{\text{sum}}(y_0)]^T \bar{y}_1$
- Obtain $[J_F]_{i,:}$ as $\bar{y}_0^T \implies$ One row at a time

Adjoint-mode algorithmic differentiation (2)

- Given $f : \mathbb{R}^N \to \mathbb{R}$ there is only one $e_i = 1$

$\Rightarrow$ Only one reverse pass computes full gradient ∇f

$\Rightarrow$ $\mathcal{O}(1)$ times primal cost

- Many names:
  - Adjoint trick, back propagation, reverse-mode AD
- Some difficulties / challenges:
  - Reverse control flow required!
  - (Hurts your heads sometimes)
  - Storage / memory costs
  - All mutation is bad ...
- One has to be a bit more clever for iterative algorithms ...