



DFTK: An algorithmically differentiable density-functional theory framework

Michael F. Herbst^{*}, Antoine Levitt[§] and Niklas Schmitz[†]

^{*}Applied and Computational Mathematics, RWTH Aachen University

[§]CERMICS, École des Ponts and Inria Paris

[†]Technische Universität Berlin

10 December 2021

Slides: <https://michael-herbst.com/slides/gdr-rest-ml>

Stress =

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



```
# 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)
)
```

Data-enhanced first-principle models

- DFT features favourable cost / accuracy ratio
- Limitations:
 - Error balancing
 - Band gap problem
 - Multi-reference situations
- Machine-learning models demonstrated predictive power
- Fraction of the cost of ground truth
- Limitations:
 - Vast amounts of data required
 - Transferability not always clear

⇒ Patch up DFT by learning additional physics from data?

Initial success stories in the literature

- 1D Kinetic energy functional¹
- Differentiable Hartree-Fock (Algopy, Diffiqult)²
- NN-based XC functional (finite differences)³
- Kohn-Sham as a regularizer (Jax, jax_dft)⁴
- NN-based XC functional (not fully self-consistent)⁵
- Arbitrary-order derivatives including CC (Jax, Quax)⁶
- NN-based XC functionals (pytorch, dqc, xcnn)⁷
- Restrictions:
 - (Mainly) Gaussian basis sets
 - Codes written from scratch
 - Build on one specific AD framework

¹J. Snyder, M. Rupp, K. Hansen, *et. al.* Phys. Rev. Lett. **108**, 253002 (2012).

²T. Tamayo-Mendoza, C. Kreisbeck, R. Lindh *et. al.* ACS Cent. Sci. **4**, 559 (2018)

³R. Nagai, A. Ryosuke and O. Sugino. npj Comp. Mater. **6**, 43 (2020)

⁴L. Li, S. Hoyer, R. Pederson *et. al.* Phys. Rev. Lett. **126**, 36401 (2021)

⁵Y. Chen, L. Zhang, H. Wang *et. al.* J. Chem. Theo. Comput. **17**, 170 (2021)

⁶A. Abbott, B. Abbott, J. Turney *et. al.* J. Phys. Chem. Lett. **12**, 3232 (2021)

⁷M. F. Kasim and S. M. Vinko. Phys. Rev. Lett. **127**, 126403 (2021)

Neural-enhanced DFT models

- DFT model defined by energy functional $E_{\text{DFT}}(\lambda, P)$
 - External parameters λ : Atomic coordinates, field, etc.
 - Density matrix P
- Variational problem of DFT:

$$\text{Given } \lambda : \min_P E_{\text{DFT}}(\lambda, P)$$

⇒ Neural-enhanced energy functional:

$$E(\theta, \lambda, P) = E_{\text{DFT}}(\lambda, P) + E_{\text{NN}}(\theta, P)$$

- Target: Optimal parameters θ given some data (e.g. $\tilde{\lambda}_i, \tilde{E}_i$)

⇒ What is optimal?

Obtaining optimal parameters θ

- Based on data $(\tilde{\lambda}_i, \tilde{E}_i)$ setup **loss function**:

$$L(\theta) = \sum_i \left| \min_P \left(E(\theta, \tilde{\lambda}_i, P) \right) - \tilde{E}_i \right|^p + \dots$$

- Now just optimise until $\nabla L = 0$, right?

Issue 1: Computing ∇L requires **unusual derivatives**

Issue 2: Physical losses may require **higher-order derivatives**

Issue 3: Dimension of θ is large, so ∇L needs to be **efficient**

Issue 1: Computing ∇L requires unusual derivatives

$$L(\theta) = \sum_i \left| \min_P (E(\theta, \tilde{\lambda}_i, P)) - \tilde{E}_i \right|^p$$

- P depends implicitly on θ (details later)
- ∇L requires $\frac{\partial P}{\partial \theta}$
- I.e. density (matrix) derivative wrt. XC/NN parameters
- Not commonly available in codes
- Associated response problem depends on XC *and* NN model

⇒ Combinatorial explosion

⇒ Manual implementation not feasible

Issue 2: Physical losses require higher-order derivatives

- Absolute energies are not physically interesting.
- **Changes** in the energy are what is interesting!
- **Properties:** How is the **response** to external **perturbation**?
- Examples:
 - Forces (response to atomic position shifts)
 - Dipole moment (response to electric field)
 - Elasticity (cross-response to lattice deformation)
 - ...

⇒ Come out as **derivatives** of the energy

⇒ Property-based losses may require **higher-order derivatives**.

Issue 3: Dimension of θ is large

$$L(\theta) = \sum_i \left| \min_P (E(\theta, \tilde{\lambda}_i, P)) - \tilde{E}_i \right|$$

- NN feature large number of parameters
- Dimension $N > 10000$ not unusual for $\theta \in \mathbb{R}^N$
- Simple finite differences

$$(\nabla L)_i \simeq \frac{L(\theta + \alpha e_i) - L(\theta)}{\alpha}$$

- Scales as $\mathcal{O}(N)$ times the cost of evaluating the **primal**
 - In this example: energy functional E , i.e. cost of an SCF

⇒ Finite-differences does not cut it

Obtaining optimal parameters θ (2)

- Based on some data $(\tilde{\lambda}_i, \tilde{E}_i)$ setup **loss function**:

$$L(\theta) = \sum_i \left| \min_P \left(E(\theta, \tilde{\lambda}_i, P) \right) - \tilde{E}_i \right|^p + \dots$$

- Now just optimise until $\nabla L = 0$, right?

Issue 1: Computing ∇L requires **unusual derivatives**

Issue 2: Physical losses may require **higher-order derivatives**

Issue 3: Dimension of θ is large, so ∇L needs to be **efficient**

- Need **efficient automated way** to obtain derivatives!

Obtaining optimal parameters θ (2)

- Based on some data $(\tilde{\lambda}_i, \tilde{E}_i)$ setup **loss function**:

$$L(\theta) = \sum_i \left| \min_P \left(E(\theta, \tilde{\lambda}_i, P) \right) - \tilde{E}_i \right|^p + \dots$$

- Now just optimise until $\nabla L = 0$, right?

Issue 1: Computing ∇L requires **unusual derivatives**

Issue 2: Physical losses may require **higher-order derivatives**

Issue 3: Dimension of θ is large, so ∇L needs to be **efficient**

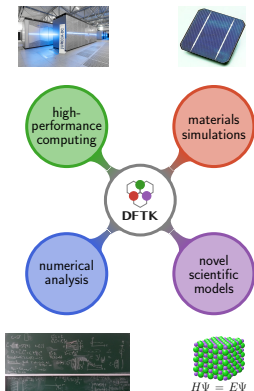
- Need **efficient automated way** to obtain derivatives!

Algorithmic differentiation (AD)

- Computational tool for computing arbitrary derivatives
- Given a *differentiable* code allows to compute derivative of
 - *any output* quantity (band gap, forces, ...) *versus*
 - *any input* (pseudo parameters, XC parameters, positions, temperature, ...)
- Adjoint-based methods cost **asymptotically the same** as function evaluation (details later)
- Usefulness goes well beyond data-enhanced DFT models:
 - Design of environment models, tight-binding, pseudopotentials
 - Sensitivity analysis & statistical inference (UQ)
 - Development of error estimates

⇒ Motivation for integration in  **DFTK**

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

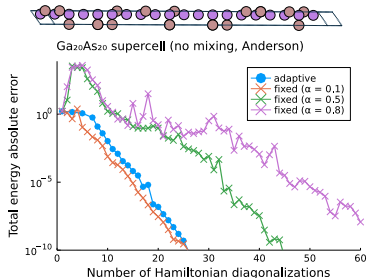
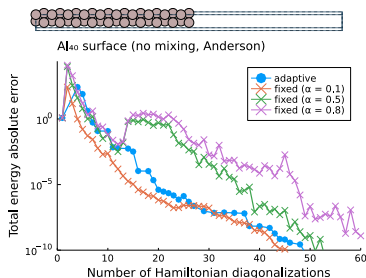


- **julia** code for plane-wave DFT
 - Fully composable with **julia** ecosystem:
 - Arbitrary precision (32bit, >64bit, ...)
 - Algorithmic differentiation
 - Numerical error control

⇒ AD capabilities are a side effect
 - Supports mathematical developments and scale-up to relevant applications
 - i.e. reduced problems for rigorous analysis (1D, analytic potentials) and DFT on > 800 electrons
- ⇒ Build with multidisciplinary research in mind
- Avoids two-language problem: Just **julia**
 - Only 2.5 years of development
 - Only 7k lines of code
- ⇒ Low entrance barrier across backgrounds

¹M. F. Herbst, A. Levitt and E. Cancès. JuliaCon Proc., 3, 69 (2021).

Black-box algorithms: Adaptive damping¹




- DFT involves a fixed-point problem, solved by SCF iteration

$$\rho^{(n+1)} = \rho^{(n)} + \alpha P^{-1} \left[\text{SCF step}(\rho^{(n)}) - \rho^{(n)} \right]$$

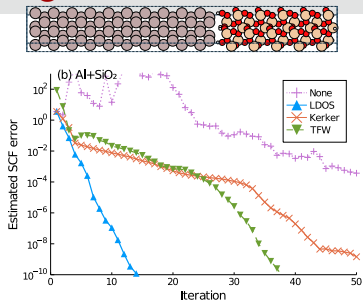
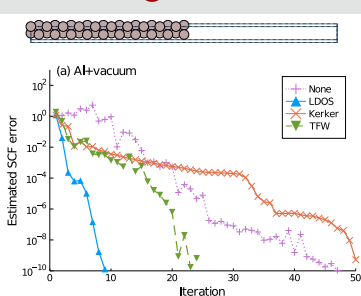
- How to choose mixing P^{-1} and damping α ?
- State-of-the-art: Guessing / trial and error (**fixed damping**)


⇒ Wasted computational time!

-  **DFTK** approach: **adaptive damping** *automatically* selects damping
- Similar performance than best fixed damping, but **fully black-box**

¹M. F. Herbst, A. Levitt. *A robust and efficient line search for self-consistent field iterations* arXiv 2109.14018.

Black-box algorithms: LDOS mixing²



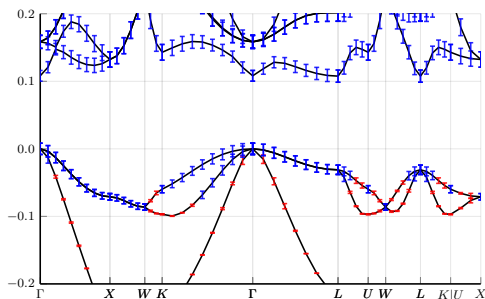
- Long-standing problem: Suitable mixing P^{-1} for inhomogeneous systems
 - E.g. metal+insulator, catalytic surfaces, ...
- State-of-the-art: local Thomas-Fermi-von Weizsäcker mixing (TFW)¹
-  DFTK approach: LDOS mixing automatically interpolates between Kerker mixing (in the metallic region) and no mixing (insulating region)

⇒ Parameter-free and black-box

¹D. Raczkowski, A. Canning, L. W. Wang, Phys. Rev. B. **64**, 121101 (2001).

²M. F. Herbst, A. Levitt. J. Phys. Condens. Matter **33**, 085503 (2021).

Rigorous error analysis: First results¹



- Fully guaranteed error bounds for band structures
- This case: Reduced Kohn-Sham model
- Captures basis set error, floating-point error, convergence error
- Recent work also considers others quantities of interest²:
- E.g. densities and forces

¹M. F. Herbst, A. Levitt and E. Cancès. Faraday Discuss. **224**, 227 (2020).

²E. Cancès, G. Dusson, G. Kemlin, A. Levitt. *Practical error bounds for properties in plane-wave electronic structure calculations* arXiv 2111.01470.

- 1 Algorithmic differentiation
- 2 Differentiating DFT simulations
- 3 Implementation in  **DFTK**
- 4 Outlook



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 $F : \mathbb{R}^2 \rightarrow \mathbb{R}$ with $F(x) = \text{double}(\text{sum}(x_1, x_2))$
- Derivative at \tilde{x} is characterised by its Jacobian matrix

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

- **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)

⇒ 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

⇒ 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 $[J_F]_{:,j} = \dot{y}_2$

⇒ Again one column of J_F at a time

- Implementation: Numbers → **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 $f : \mathbb{R}^N \rightarrow \mathbb{R}$ is a differentiable function, computing $\nabla f = J_f$ is asymptotically not more expensive than f itself.

⇒ 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)

$$F(x) = b^T L x + \mathcal{O}(x^2)$$

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)

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

\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:*
 - ➊ Set $y_0 = (x_1, x_2)$
 - ➋ Compute $y_1 = \text{sum}(y_0)$ and store it
 - ➌ Compute $y_2 = \text{double}(y_1)$ and store it
- *Reverse pass:*
 - ➊ Set $\bar{y}_2 = e_i$
 - ➋ Compute $\bar{y}_1 = [J_{\text{double}}(y_1)]^T \bar{y}_2 \longleftarrow$
 - ➌ Compute $\bar{y}_0 = [J_{\text{sum}}(y_0)]^T \bar{y}_1 \longleftarrow$
- Obtain $[J_F]_{i,:}$ as $\bar{y}_0^T \implies$ One **row** at a time

Adjoint-mode algorithmic differentiation (2)

- Given $f : \mathbb{R}^N \rightarrow \mathbb{R}$ 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 ...
 - Let's look at the SCF case next.

Contents

- 1 Algorithmic differentiation
- 2 Differentiating DFT simulations
- 3 Implementation in  **DFTK**
- 4 Outlook

Properties and derivatives of SCFs

- SCF fixed-point problem in density matrix P

$$0 = f(P, \lambda) = f_{\text{FD}}\left(H^\lambda(P)\right) - P$$

with

- λ : Parameters (*for simplicity*: both external & neural net)
- f_{FD} : Fermi-Dirac function
- H^λ : Non-linear Kohn-Sham Hamiltonian
- Defines implicit function $P(\lambda)$ for density matrix
- Quantities of interest:

$$\frac{dA(P)}{d\lambda} = \frac{\partial A}{\partial \lambda} + \frac{\partial A}{\partial P} \frac{\partial P}{\partial \lambda}$$

- Forces: $A = E$, $\lambda = R$ (atomic displacement)
- Polarisability: $A = \text{dipole moment}$, $\lambda = \mathcal{E}$ (electric field)

Hellmann-Feynman theorem

$$\frac{dA(P)}{d\lambda} = \frac{\partial A}{\partial \lambda} + \frac{\partial A}{\partial P} \frac{\partial P}{\partial \lambda}$$

- Special case of $A = E$
- Recall $P_* = \operatorname{argmin} E(P) \Rightarrow \left. \frac{\partial E}{\partial P} \right|_{P_*} = 0$
- Hellmann-Feynman theorem

$$\left. \frac{dE}{d\lambda} \right|_* = \left. \frac{\partial E}{\partial \lambda} \right|_*$$

- First energy derivatives are (comparatively) easy!

Response theory (1)

- If $A \neq E$ we need $\frac{\partial P}{\partial \lambda}$
- Consider at $\lambda = \lambda_*$ and corresponding P_* and H_* :

$$\begin{aligned} 0 &= \frac{\partial}{\partial \lambda} \left[f_{\text{FD}}(H^\lambda(P)) - P \right] \Big|_* \\ &= f'_{\text{FD}}(H_*) \cdot \frac{\partial H^\lambda}{\partial \lambda} \Big|_* + \frac{\partial P}{\partial \lambda} \Big|_* \cdot \frac{\partial}{\partial P} \left[f_{\text{FD}}(H^\lambda(P)) - P \right] \Big|_* \\ &= f'_{\text{FD}}(H_*) \cdot \frac{\partial H^\lambda}{\partial \lambda} \Big|_* + \frac{\partial P}{\partial \lambda} \Big|_* \cdot \left[f'_{\text{FD}}(H_*) \cdot \mathbf{K}^{\lambda_*}(P_*) - I \right] \\ &= \chi_0(H_*) \cdot \frac{\partial H^\lambda}{\partial \lambda} \Big|_* - \frac{\partial P}{\partial \lambda} \Big|_* \cdot \left[I - \chi_0(H_*) \cdot \mathbf{K}^{\lambda_*}(P_*) \right] \end{aligned}$$

$$\text{where } \mathbf{K}^{\lambda_*} = \frac{\partial H^{\lambda_*}}{\partial P}, \chi_0(H_*) = f'_{\text{FD}}(H_*)$$

Response theory (2): Sternheimer equation

$$0 = \chi_0(H_*) \cdot \left. \frac{\partial H^\lambda}{\partial \lambda} \right|_* - \left. \frac{\partial P}{\partial \lambda} \right|_* \cdot [I - \chi_0(H_*) \cdot \mathbf{K}^{\lambda*}(P_*)]$$

- Rearrange:

$$\begin{aligned} \left. \frac{\partial P}{\partial \lambda} \right|_* &= [I - \chi_0(H_*) \mathbf{K}^{\lambda*}(P_*)]^{-1} \chi_0 \left. \frac{\partial H^\lambda}{\partial \lambda} \right|_* \\ &= - [\mathbf{K}^{\lambda*}(P_*) + \mathbf{\Omega}(H_*)]^{-1} \left. \frac{\partial H^\lambda}{\partial \lambda} \right|_* \end{aligned}$$

$$\text{where } \mathbf{\Omega}(H_*) = -[\chi_0(H_*)]^{-1}$$

- Sternheimer equation (implicit differentiation)

Example: Computing polarisabilities

- Homogeneous electric field $\lambda = \mathcal{E}$ along x -direction
- Cubic cell (length L_x)
- Hamiltonian $H^\mathcal{E}(P) = H_{\text{DFT}}(P) - \mathcal{E}(x - L_x/2)$
- Perturbation $\left. \frac{\partial H^\mathcal{E}}{\partial \mathcal{E}} \right|_* = (x - L_x/2)$
- Dipole moment:

$$\mu(P) = \int_{\Omega} (x - L_x/2) \rho(r) \, dr, \quad \rho = \text{diag}(P)$$

- Polarisability $\frac{d\mu}{d\mathcal{E}} = \frac{\partial \mu}{\partial P} \frac{\partial P}{\partial \mathcal{E}}$
- 1 Solve SCF $P_* = H^0(P_*)$ at zero field
 - 2 Solve Sternheimer $\frac{\partial P}{\partial \mathcal{E}} = -[\mathbf{K} + \mathbf{\Omega}]^{-1} \frac{\partial H^\mathcal{E}}{\partial \mathcal{E}}$ (*implicit differentiation*)
 - 3 Compute polarisability

Role of algorithmic differentiation

- For electronic structure theory:

- SCF is a *frequent primitive*

- Code up SCF and Sternheimer for AD library **once**

⇒ AD library can invoke it as needed

⇒ User asks for arbitrary gradient, appropriate response problem solved **automatically**

- Adjoint-mode differentiation:

- $\mathbf{K} + \mathbf{\Omega}$ is self-adjoint (i.e. one solver for both modes)

⇒ One Sternheimer solve per *output* parameter

- E.g. for energy quantities **one solve** for **all sensitivities**


⇒ Supports large number of parameters & NN-based approaches

- Additional goodies

- E.g. support for higher derivatives, sparsification techniques

Contents

- 1 Algorithmic differentiation
- 2 Differentiating DFT simulations
- 3 Implementation in  **DFTK**
- 4 Outlook

- Time investment:
 - Bachelor student, $\simeq 12$ weeks half-time (20h/week)
 - Some follow-up work and support from  DFTK developers
- Forward-mode status (`ForwardDiff.jl`):
 - DFT fully supported
 - Some polishing in user interface needed
 - Default approach for stresses
- Adjoint-mode status (`Zygote.jl`):
 - `ChainRules.jl`: **No hard commitment** to a single AD tool
 - Limited to reduced models
 - Difficulties: Third-party C codes, program flow & mutation
 - Ongoing work ...

Forward-mode AD with Hellman-Feynman

- For stresses $A = E$, $\lambda = L$ (unit cell vectors)

⇒ Hellmann-Feynman applies

- Computing stresses:

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

- In  code¹

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

¹Live code: <https://github.com/JuliaMolSim/DFTK.jl/blob/master/src/postprocess/stresses.jl>

DEMO

Polarisabilities by algorithmic differentiation

<https://docs.dftk.org/stable/examples/forwarddiff/>

Contents

- 1 Algorithmic differentiation
- 2 Differentiating DFT simulations
- 3 Implementation in  DFTK
- 4 Outlook




Conclusion

- **Data-enhanced methods**: Need for unusual gradients
- Exploration of such methods has just started

⇒ Flexible differentiable codes are key

- Size of parameter space requires **adjoint-mode AD**
- Practical challenges (program runs in reverse!)

⇒ Best framework not clear

-  **DFTK**: **Initial support** for forward & reverse AD
- Profit from composable  ecosystem:
 - **AD is a side effect** ( **DFTK** not written for AD)
 - No full buy-in into a single AD framework.
- We are not ML people: Happy for any input!

Opportunities to learn more ...

RWTH  workshop:

“Introduction to the Julia programming language”


- **17th and 18th February 2022** (online & open for everyone)

⇒ <https://michael-herbst.com/learn-julia>



DFTK school 2022 (*with E. Cancès, A. Levitt*):

“Numerical methods for DFT simulations”

- **29–31 August 2022** at Sorbonne Université, Paris
- Centred around  **DFTK** and its multidisciplinary philosophy
- Grounds-up introduction of electronic structure theory, mathematical background, numerical methods, implementation
- Applications in method development & simulations

⇒ <https://school2022.dftk.org>

Antoine Levitt

Niklas Schmitz

Benjamin Stamm

Eric Cancès

all DFTK contributors



Summer of code





DFTK <https://dftk.org>,
<https://school2022.dftk.org>



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



mfherbst



<https://michael-herbst.com/blog>



herbst@acom.rwth-aachen.de