Sparse Automatic Differentiation

The fastest Jacobians in the West

Guillaume Dalle
Alexis Montoison
Adrian Hill

EPFL, INDY lab seminar

28.11.2024

Motivation

- Many algorithms require derivative matrices:
  - Differential equations $\to$ Jacobian
  - Convex optimization $\to$ Hessian
- Large matrices are expensive to compute and store…
- … except when they are sparse!
- Sparsity can be leveraged automatically.

What is AD?

input	**output**
program to compute the function $x \longmapsto f(x)$	program to compute the derivative $x \longmapsto \partial f(x)$

Two ingredients only:

1. hardcode basic derivatives ($+$, $\times$, exp, log, …)
2. handle compositions $f = g \circ h$

Linear operators

For a function $f = g \circ h$, the chain rule gives

$$\begin{aligned} \text{standard} \qquad & \partial f(x) = \partial g(h(x)) \circ \partial h(x) \\ \text{adjoint} \qquad & \partial f(x)^* = \partial h(x)^* \circ \partial g(h(x))^* \end{aligned}$$

These linear operators apply as follows:

$$\begin{aligned} \text{forward} \qquad & \partial f(x) : u \xmapsto{\partial h(x)} v \xmapsto{\partial g(h(x))} w \\ \text{reverse} \qquad & \partial f(x)^* : u \xleftarrow[\partial h(x)^*]{} v \xleftarrow[\partial g(h(x))^*]{} w \end{aligned}$$

Two modes

Forward-mode AD computes Jacobian-Vector Products:

$$u \longmapsto \partial f(x)[u]$$

Reverse-mode AD computes Vector-Jacobian Products:

$$w \longmapsto \quad \partial f(x)^*[w] = w^* \partial f(x)$$

No need to materialize intermediate Jacobian matrices!

Cost of 1 JVP or VJP for $f \propto$ cost of 1 evaluation of f.

From operators to matrices

To compute the Jacobian matrix J of a composition $f : \mathbb{R}^m \longrightarrow \mathbb{R}^n$:

- ~~product of intermediate Jacobian matrices~~
- reconstruction from JVPs or VJPs

	forward mode	**reverse mode**
idea	1 JVP gives 1 column	1 VJP gives 1 row
formula	$J_{\cdot,j} = \partial f(x)[e_j]$	$J_{i,\cdot} = \partial f(x)^*[e_i]$
cost	n JVPs (input dimension)	m JVPs (output dimension)

Using fewer products

When the Jacobian is sparse, we can compute it faster.

If columns $j_1, \ldots, j_k$ of J are structurally orthogonal (their nonzeros never overlap), we deduce them all from a single JVP:

$$J_{j_1} + \ldots + J_{j_k} = \partial f(x)\left[e_{j_1} + \ldots + e_{j_k}\right]$$

Once we have grouped columns, sparse AD has two steps:

1. compressed differentiation of each group $c = \{j_1, \ldots, j_k\}$
2. decompression into individual columns $j_1, \ldots, j_k$

Compression

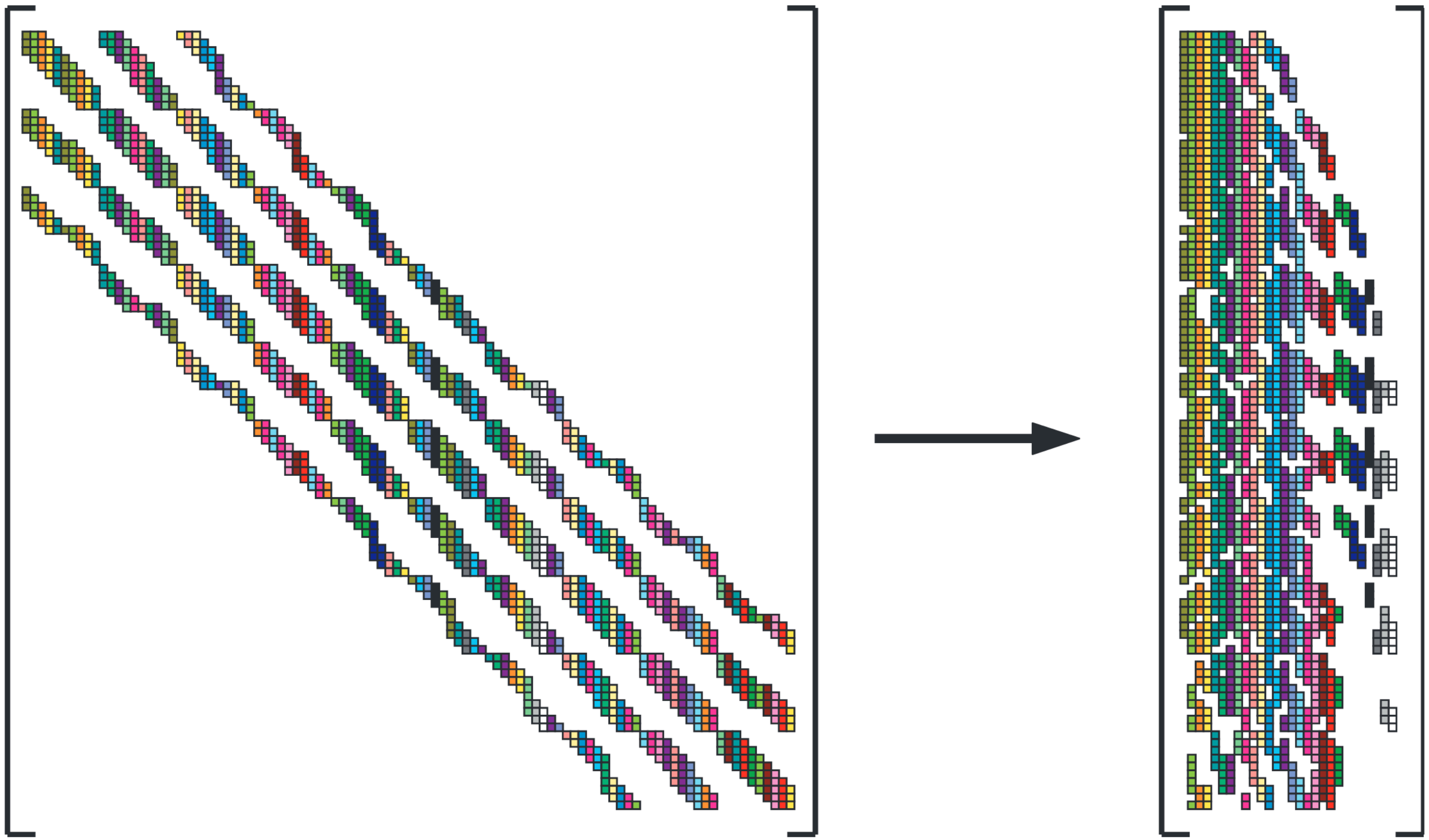


Figure 1: Gebremedhin et al. (2005)

Two preliminary steps

When grouping columns, we want to

- guarantee structural orthogonality (correctness)
- form the smallest number of groups (efficiency)

preparation	**execution**
1. structure detection 2. coloring	3. compressed differentiation 4. decompression

The preparation phase can be amortized across several inputs.

Partitions for matrix A

Orthogonal for all (i, j) s.t. $A_{ij} \neq 0$,

- column j is alone in group $c(j)$ with a nonzero in row i

Symmetrically orthogonal for all (i, j) s.t. $A_{ij} \neq 0$,

- either column j is alone in group $c(j)$ with a nonzero in row i
- or column i is alone in group $c(i)$ with a nonzero in row j

Each partition can be reformulated as a specific coloring problem.

Graph types for matrix A

Column intersection $(j_1, j_2) \in \mathcal{E} \iff \exists i, A_{ij_1} \neq 0$ and $A_{ij_2} \neq 0$

Bipartite $(i, j) \in \mathcal{E} \iff A_{ij} \neq 0$ (2 vertex sets $\mathcal{I}$ and $\mathcal{J}$)

Adjacency (sym.) $(i, j) \in \mathcal{E} \iff i \neq j \;\&\; A_{ij} \neq 0$

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 & 0 & a_{16} & a_{17} & a_{18} \\ 0 & a_{22} & 0 & 0 & a_{25} & 0 & a_{27} & a_{28} \\ 0 & 0 & a_{33} & 0 & a_{35} & a_{36} & 0 & a_{38} \\ 0 & 0 & 0 & a_{44} & a_{45} & a_{46} & a_{47} & 0 \end{bmatrix}$$

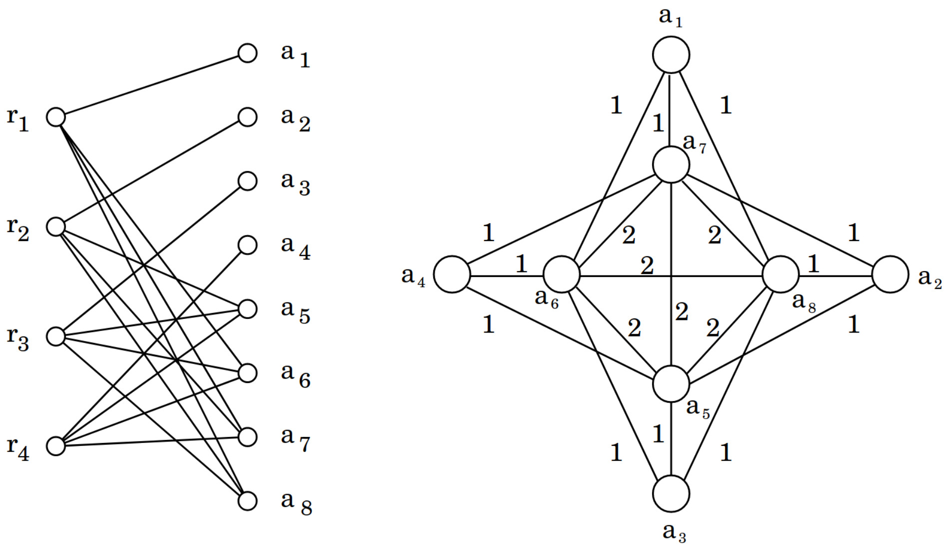


Figure 2: Gebremedhin et al. (2005)

Jacobian coloring

Coloring of intersection graph / distance-2 coloring of bipartite graph

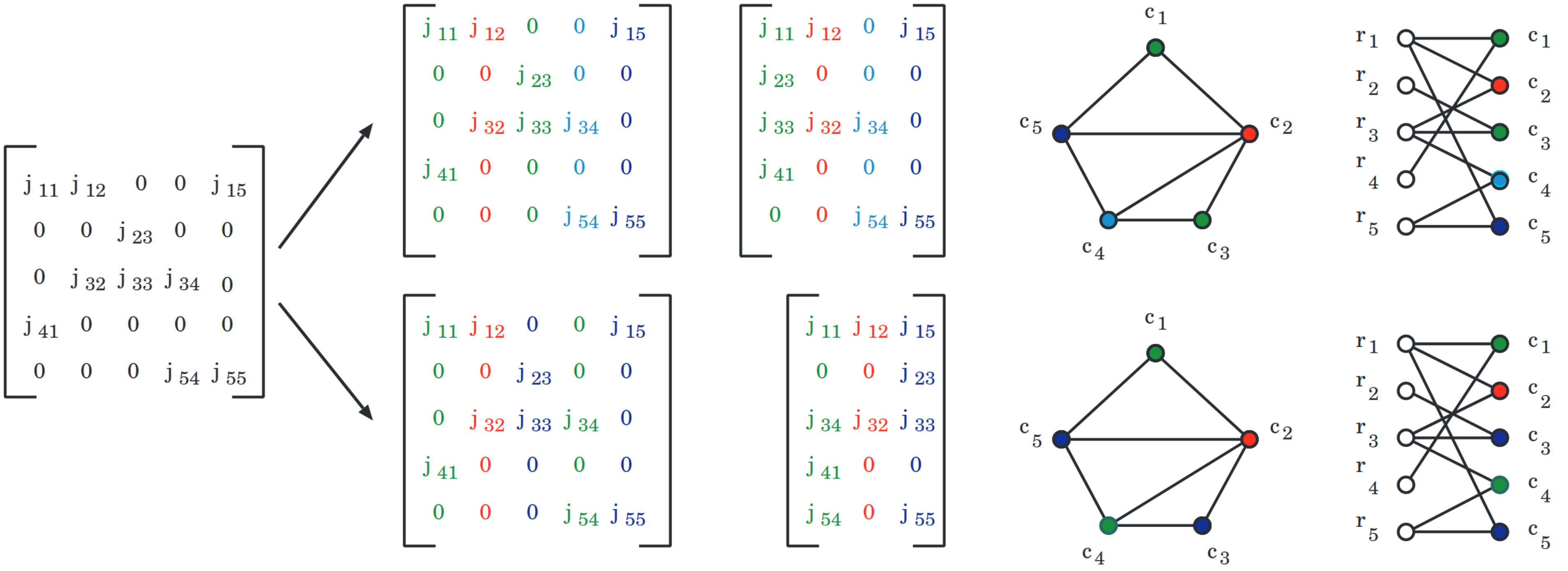


Figure 3: Gebremedhin et al. (2005)

Hessian coloring

Star coloring of adjacency graph

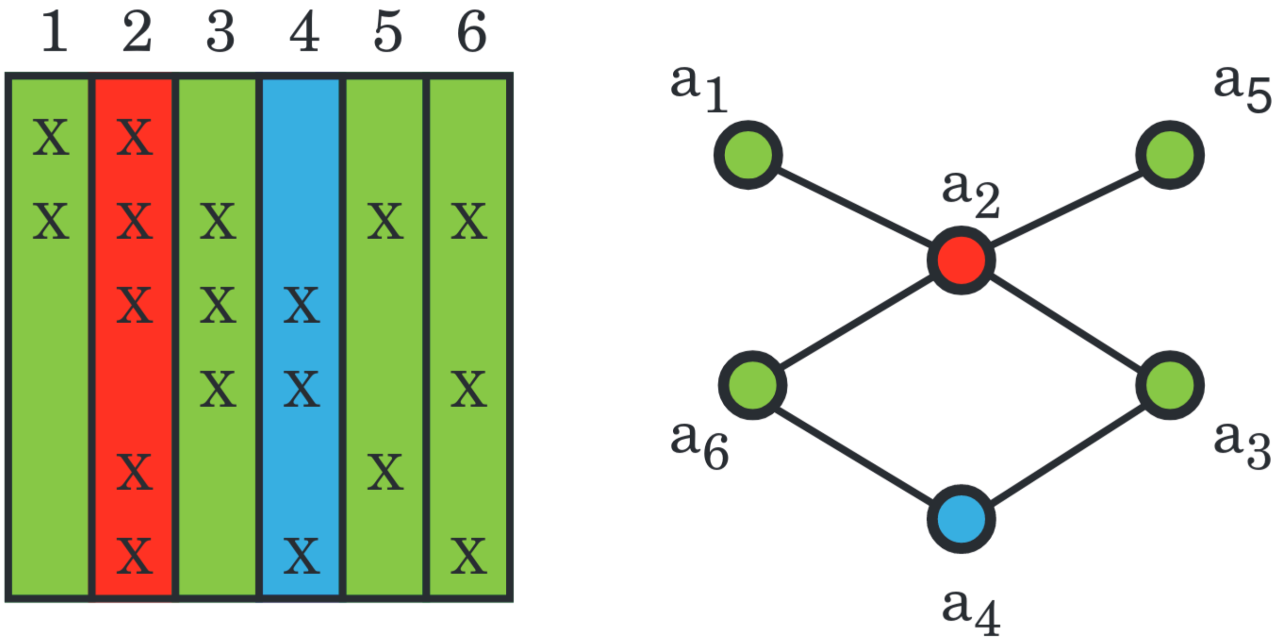


Figure 4: Gebremedhin et al. (2005)

Hessian coloring (2)

Why a "star" coloring? Consider

$$A = \begin{pmatrix} A_{kk} & A_{ki} & \cdot & \cdot \\ A_{ik} & A_{ii} & A_{ij} & \cdot \\ \cdot & A_{ji} & A_{jj} & A_{jl} \\ \cdot & \cdot & A_{lj} & A_{ll} \end{pmatrix}$$

If coloring c yields a symmetrically orthogonal partition:

- $c(i) \neq c(j)$
- $c(i) \neq c(k)$
- $c(j) \neq c(l)$

Any path on 4 vertices must use at least 3 colors $\Longleftrightarrow$ any 2-colored subgraph is a collection of disjoint stars.

Jacobian bicoloring

Bidirectional coloring of bipartite graph, with neutral color

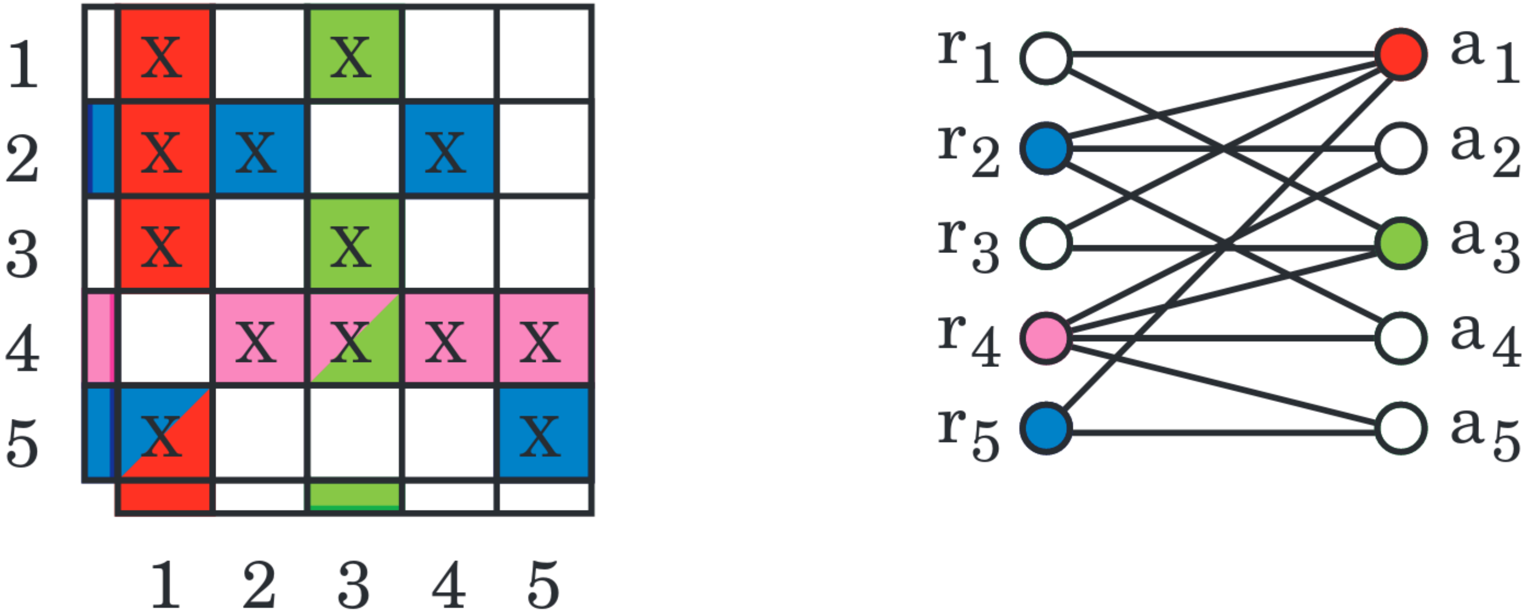


Figure 5: Gebremedhin et al. (2005)

New insights on Jacobian bicoloring

1. A bicoloring of J is given by a star coloring of $H = \begin{pmatrix} 0 & J \\ J^T & 0 \end{pmatrix}$: we can reuse existing algorithms
2. Diagonal of H is now all zero: we can relax star coloring into *no zig-zag coloring*

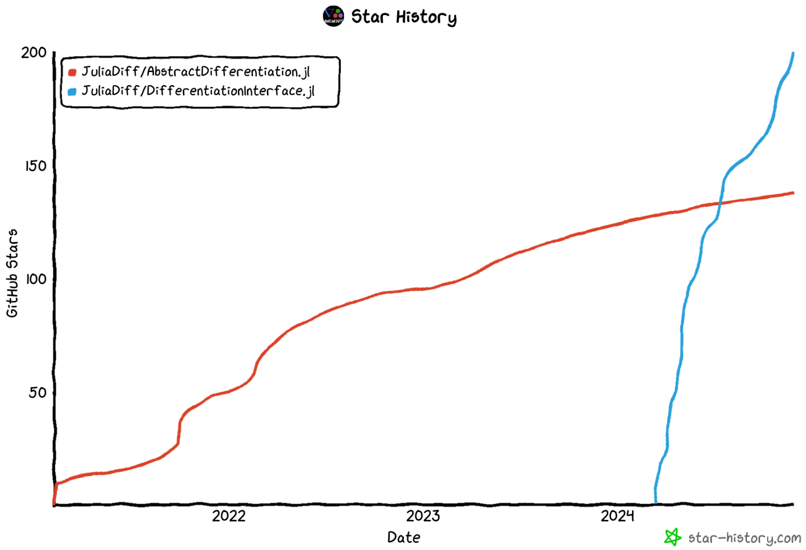
$$A = \begin{pmatrix} \cdot & A_{ki} & \cdot & \cdot \\ A_{ik} & \cdot & A_{ij} & \cdot \\ \cdot & A_{ji} & \cdot & A_{jl} \\ \cdot & \cdot & A_{lj} & \cdot \end{pmatrix}$$

No path on 4 vertices can have colors (c_1, c_2, c_1, c_2).

A high-level sparse AD ecosystem

Independent packages working together:

- Step 1 (structure detection): `SparseConnectivityTracer.jl`
- Steps 2 & 4 (coloring, decompression): `SparseMatrixColorings.jl`
- Step 3 (compressed diffeentiation): `Differentiationterface.jl`



Impact

	`SCT.jl`	`SMC.jl`	`DI.jl`
lines of code	4719	3600	14332
indirect dependents	387	350	345
downloads / month	10k	17k	19k

Users already include…

- Scientific computing: SciML (Julia's `scipy`)
  - Differential equations, nonlinear solves, optimization
- Probabilistic programming: `Turing.jl`
- Symbolic regression: `PySR`

Perspectives

- GPU-compatible structure detection and coloring
- Pure Julia autodiff engine based on SSA-IR (`Mooncake.jl`)

Going further

On general AD:

- Blondel & Roulet (2024)
- Margossian (2019)
- Baydin et al. (2018)

On sparse AD:

- Griewank & Walther (2008)
- Gebremedhin et al. (2005)
- Walther (2008)

Bibliography

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*, *18*(153), 1–43. http://jmlr.org/papers/v18/17-468.html

Blondel, M., & Roulet, V. (2024, July). *The Elements of Differentiable Programming*. arXiv. https://doi.org/10.48550/arXiv.2403.14606

Gebremedhin, A. H., Manne, F., & Pothen, A. (2005). What Color Is Your Jacobian? Graph Coloring for Computing Derivatives. *SIAM Review*, *47*(4), 629–705. https://doi.org/10/cmwds4

Bibliography

Griewank, A., & Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation* (2nd ed). Society for Industrial, Applied Mathematics.

Margossian, C. C. (2019). A review of automatic differentiation and its efficient implementation. *Wires Data Mining and Knowledge Discovery*, *9*(4), e1305. https://doi.org/10.1002/widm.1305

Walther, A. (2008). Computing sparse Hessians with automatic differentiation. *ACM Transactions on Mathematical Software*, *34*(1), 1–15. https://doi.org/10.1145/1322436.1322439