Computer Physics Communications 309 (2025) 109491

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computational Physics

Optimization using pathwise algorithmic derivatives of electromagnetic shower simulations ☆

Max Aehle [a,*], Mihály Novák [b], Vassil Vassilev [c], Nicolas R. Gauger [a], Lukas Heinrich [d], Michael Kagan [e], David Lange [c]

[a] University of Kaiserslautern-Landau (RPTU), Gottlieb-Daimler-Straße, 67663 Kaiserslautern, Germany
[b] European Organization for Nuclear Research (CERN), Esplanade des Particules 1, 1217 Meyrin, Geneva, Switzerland
[c] Princeton University, Department of Physics, Jadwin Hall, Washington Road, Princeton, NJ 08544-0708, USA
[d] TU Munich, Arcisstraße 21, 80333 Munich, Germany
[e] SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA 94025-7015, USA

ARTICLE INFO

Keywords:
Automatic differentiation
Differentiable programming
Gradient estimation
Monte-Carlo algorithm
High-energy physics
Sampling calorimeter

ABSTRACT

Among the well-known methods to approximate derivatives of expectancies computed by Monte-Carlo simulations, averages of pathwise derivatives are often the easiest one to apply. Computing them via algorithmic differentiation typically does not require major manual analysis and rewriting of the code, even for very complex programs like simulations of particle-detector interactions in high-energy physics. However, the pathwise derivative estimator can be biased if there are discontinuities in the program, which may diminish its value for applications.

This work integrates algorithmic differentiation into the electromagnetic shower simulation code HepEmShow based on G4HepEm, allowing us to study how well pathwise derivatives approximate derivatives of energy depositions in a sampling calorimeter with respect to parameters of the beam and geometry. We found that when multiple scattering is disabled in the simulation, means of pathwise derivatives converge quickly to their expected values, and these are close to the actual derivatives of the energy deposition. Additionally, we demonstrate the applicability of this novel gradient estimator for stochastic gradient-based optimization in a model example.

1. Introduction

Monte-Carlo simulations. Monte-Carlo (MC) simulations are a popular method to model processes that involve stochasticity; for instance, the Geant4 toolkit [1–3] is widely used to simulate the passage of particles through matter. Unlike deterministic simulations, the output data $y \in Y \subset \mathbb{R}^m$ of MC simulations does not only depend on the input data $\theta \in \Theta \subset \mathbb{R}^n$, but also on random numbers supplied by a pseudo-random number generator (RNG). We can think of MC simulations as functions

$$f : \Theta \times \Omega \to Y, (\theta, \omega) \mapsto y \tag{1}$$

with an additional argument ω from a probability space Ω with a probability measure $\mathbb{P}$. For simplicity, we assume in the following that the RNG defines only a single stochastic primitive called `flat()` (as in Geant4) that returns independent random numbers uniformly distributed on the interval $[0, 1]$, like `numpy.random.rand` in Python or `(double)rand()/RAND_MAX` in C. We may think of Ω as the set of sequences of random numbers.

Usually, the function f is evaluated many times; a common quantity of interest for a MC simulation is the expected value of the output for a given input θ,

$$\mathbb{E}f := \mathbb{E}_\omega f(\theta, \omega) = \int f(\theta, \omega)\, \mathrm{d}\mathbb{P}(\omega), \tag{2}$$

which can be estimated by averaging over N independent random samples,

$$\overline{f} := \frac{1}{N} \cdot \sum_{i=1}^{N} f(\theta, \omega^{(i)}). \tag{3}$$

☆ The review of this paper was arranged by Prof. David W. Walker.

* Corresponding author.
E-mail addresses: max.aehle@scicomp.uni-kl.de (M. Aehle), mihaly.novak@cern.ch (M. Novák), vassil.vassilev@cern.ch (V. Vassilev), nicolas.gauger@scicomp.uni-kl.de (N.R. Gauger), l.heinrich@tum.de (L. Heinrich), makagan@slac.stanford.edu (M. Kagan), david.lange@cern.ch (D. Lange).

https://doi.org/10.1016/j.cpc.2024.109491
Received 18 June 2024; Received in revised form 13 December 2024; Accepted 23 December 2024

Available online 30 December 2024

The choice of N must balance the required run-time, which grows linearly with N, with the standard deviation of $\overline{f}$, which is proportional to $N^{-1/2}$.

Algorithmic Differentiation. Sometimes, users of (for now, deterministic) computer simulations are not primarily interested in the output y at a specific input value θ, but rather wish to identify *optimal inputs* $\theta \in \Theta \subset \mathbb{R}^n$ that maximize or minimize a scalar output y. For example, θ might be a set of parameters to be tuned in order to minimize the deviation y between model predictions and observed data; closely related, θ might contain the weights of a neutral network and y be the training error. As another example, θ might be a set of design parameters and y a utility function to be improved, see e.g. the work of Albring et al. [4] who optimized the shape of an airfoil to reduce the drag computed by a computational fluid dynamics simulation.

To employ *gradient-based optimization* methods like gradient descent or BFGS [5], it is necessary to be able to evaluate gradients $\partial y/\partial\theta$. Besides, the derivative $\partial y/\partial\theta$ characterizes the sensitivity of y with respect to changes in θ, and can thus be useful for uncertainty quantification. When the function $\theta \mapsto y$ is given by computer code, the gradient $\partial y/\partial\theta$ of a computer-implemented deterministic function can often be obtained efficiently and accurately by *algorithmic differentiation (AD)* [6], a set of techniques based on the chain rule and the well-known derivatives of the elementary operations performed by the computer program while evaluating $\theta \mapsto y$.

Specifically, the *forward mode* of AD with a single scalar input $\theta \in \mathbb{R}$ (i.e. $n = 1$) keeps track of the *dot value* $\dot{q} = \partial q/\partial\theta$ whenever an intermediate variable q is computed by the program; for example, the *primal* operation $q = q_1 \cdot q_2$ is augmented with the *AD logic* $\dot{q} = \dot{q}_1 \cdot q_2 + q_1 \cdot \dot{q}_2$. Optimization applications favor the *reverse mode* of AD because, unlike the forward mode of AD and unlike difference quotients, it provides the entire gradient of a scalar output $y \in \mathbb{R}$ (i.e. $m = 1$) with respect to many inputs $\theta \in \mathbb{R}^n$ in a run-time independent from n, however at the expense of a higher memory consumption; we refer to the textbook by Griewank and Walther [6] for details. On the implementation side, there are several mechanisms for *AD tools* to detect real arithmetic in an existing primal program and to augment them with AD logic; for instance, *operator-overloading tools* provide a custom floating-point datatype with arithmetic operators and math functions overloads, to be used instead of the built-in floating-point types like `double` in C++. In contrast, *source transformation* tools operate on the program as a whole. While they are usually more difficult to implement and may only support a subset of the language, having access to the entire program allows for more advanced performance optimizations.

AD for MC Simulations. Typically, AD tools recognize and differentiate the basic operations $+$, $-$, $\cdot$ and $/$ and related operators like $+=$, as well as simple math functions like $\sqrt{\ }$, exp, sin, etc. Higher-level mathematical constructs often need manual treatment; while it is usually straightforward to inform AD with analytical derivatives of, e.g., solutions of linear systems [7] and the dominating eigenvalue of a matrix [8], computing the derivative of an expected value of a MC simulation,

$$(\mathbb{E}f)' := \frac{\partial}{\partial\theta}\left[\mathbb{E}_\omega f(\theta,\omega)\right] = \frac{\partial}{\partial\theta}\left[\int f(\theta,\omega)\,\mathrm{d}\mathbb{P}(\omega)\right], \tag{4}$$

poses a rather difficult but very important challenge across application domains.

In quantitative finance, certain derivatives of e.g. expected option prices are called "Greeks" and define strategies to hedge risks [9]. Differentiable rendering allows to reconstruct three-dimensional scenes from images [10–14]. In reinforcement learning, policy gradients can be used for training [15]. In many of the aforementioned application areas of gradient-based optimization using deterministic AD, it is natural to add stochasticity to the differentiated code, leading to e.g. stochastic neural networks [16] including VAEs [17] and GANs [18]. See [19] for a review of Monte Carlo gradient estimation in machine learning.

The present work is a study on applying AD in the realm of high-energy physics (HEP), where gradient-based optimization is explored as a way to enhance the design of future particle detectors [20,21] or reconstruct properties of detected particles [22,23], and gradients of stochastic programs could help performing Bayesian inference of parameters of the standard model [24]. The Geant4 toolkit for the simulation of the passage of particles through matter [1–3] is widely used across many HEP-related application areas, from the planning of detectors at the LHC to radiation safety in space to medical physics.

As a step to explore ways to create a differentiated version of Geant4, in this study, we differentiate a more compact but algorithmically similar MC code composed of G4HepEm [25] and HepEmShow [26,27]. We are interested in derivatives of the expected value of energy deposition of electromagnetic showers in a simple sampling calorimeter, with respect to parameters of the geometry and the incoming particles.

A natural first step to approach (4) is to form the *pathwise derivative*

$$\frac{\partial}{\partial\theta} f(\theta,\omega) \tag{5}$$

by applying AD to the MC simulation f in a way that, with regard to differentiation, treats random numbers like constants. This has been accomplished for Geant4 in principle, without focus on performance though and only simulating a single particle to demonstrate technical feasibility [21]. The second step then is to estimate the expected value of the pathwise derivative,

$$\mathbb{E}(f') := \mathbb{E}_\omega\left[\frac{\partial}{\partial\theta} f(\theta,\omega)\right], \tag{6}$$

by averaging it over N_{diff} independent random samples,

$$\overline{f'} := \frac{1}{N_{\mathrm{diff}}} \cdot \sum_{i=1}^{N_{\mathrm{diff}}} \left[\frac{\partial}{\partial\theta} f(\theta,\omega^{(i)})\right]. \tag{7}$$

However, the expected pathwise derivative $\mathbb{E}(f')$ matches the sought derivative $(\mathbb{E}f)'$ of the expected value only under certain assumptions on f. A well-known corollary of Lebesgue's dominated convergence theorem [28, Theorem A.5.3] states that $(\mathbb{E}f)' = \mathbb{E}(f')$ if $f(\theta,\omega)$ is continuously differentiable in θ and $|\frac{\partial f}{\partial\theta}| \le B(\omega)$ for an integrable random variable $B : \Omega \to \mathbb{R}$. Fig. 1 gives an example of such a function

$$f_1(\theta,\omega) = \left\{\begin{array}{ll} 1, & r(\omega) < 0.6 \\ 2, & r(\omega) \ge 0.6 \end{array}\right\} + \theta \cdot (d + \sin(8\pi r(\omega))) \tag{8}$$

with $(\mathbb{E}f_1)' = \mathbb{E}(f_1') = d$, where $r(\omega)$ is a random variable uniformly distributed on $[0,1]$.

In general, $\mathbb{E}(f')$ and $(\mathbb{E}f)'$ can take different values. For the function

$$f_2(\theta,\omega) = \begin{cases} 1, & r(\omega) < 0.6 - d\cdot\theta \\ 2, & r(\omega) \ge 0.6 - d\cdot\theta \end{cases} \tag{9}$$

in Fig. 1, we can analytically see that

$$\mathbb{E}_\omega f_2(\theta,\omega) = (0.6 - d\cdot\theta)\cdot 1 + (0.4 + d\cdot\theta)\cdot 2 = 1.4 + d\cdot\theta$$
$$\Rightarrow \frac{\partial}{\partial\theta}\left[\mathbb{E}_\omega f_2(\theta,\omega)\right] = d,$$

but the pathwise derivative $\frac{\partial}{\partial\theta} f_2(\theta,\omega)$ is zero for almost all ω. Only for the zero-probability set of ω with $r(\omega) = 0.6 - d\cdot\theta$, $f_2(\theta,\omega)$ has a jump at θ. This jump makes $(\mathbb{E}f)'$ non-zero but does not affect $\mathbb{E}(f')$. An estimator like $\overline{f'}$, whose expected value does not match the target value $(\mathbb{E}f)'$, is called *biased*.

Non-trivial MC simulations usually contain control flow constructs like `if` and `while`, whose (discrete and hence non-differentiable) condition depends on both the AD inputs θ and the randomness ω, so their pathwise derivative estimators are generally biased. Accordingly, several approaches to create unbiased estimates for derivatives of expected values of MC simulations have been proposed in the literature; see e.g. references [29,30,13], or Kagan and Heinrich [31] for a first analysis of some of these methods in HEP.

The *reparametrization trick* [17] refers to implementing parametric random distributions as differentiable expressions of the parameters and

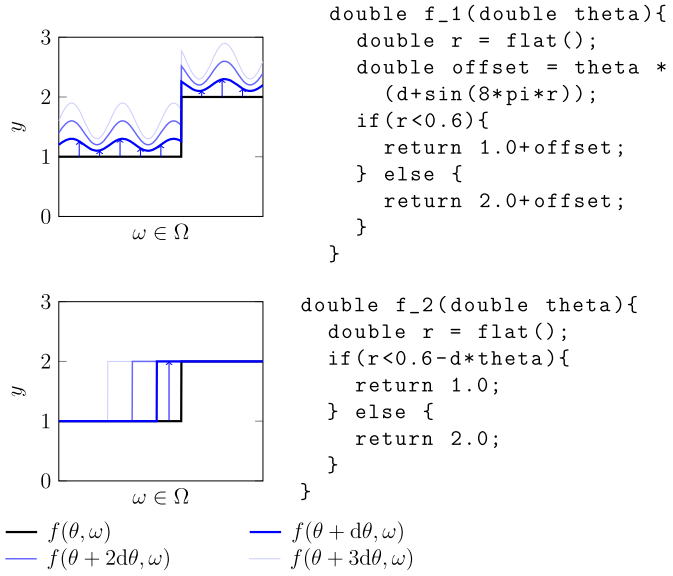

Fig. 1. Different mechanisms for Monte-Carlo simulations to combine input parameters and randomness. The RNG primitive `flat()` yields independent random numbers uniformly distributed on $[0,1]$; this is how the second argument ω in (1) comes in. Around $\theta = 0$, both functions have $\mathbb{E}_\omega f_i(\theta,\omega) = d \cdot \theta$. Top: The pathwise derivative is distributed around the mean d. Bottom: The pathwise derivative is zero almost everywhere, and undefined at a single point where f jumps.

non-parametric random numbers. For instance, a random number uniformly distributed on $[0,\theta]$ (with $0 < \theta \leq 1$, say) would be implemented as $\theta \cdot$ `flat()` rather than, e.g., rejection-sampling `flat()` repeatedly until it yields a number in $[0,\theta]$. This makes the differentiable dependency between the sampled random numbers and the parameters visible to the mean pathwise derivative $\mathbb{E}(f')$. For elementary parametric distributions like normal distributions, MC simulations in HEP often follow the reparametrization trick by default. However, MC simulations then usually continue to process them using non-differentiable operations (like f_2 in Fig. 1), to which the reparametrization trick cannot be applied in general.

Christodoulou and Naumann [32] and Kreikemeyer and Andelfinger [33] have presented frameworks to smoothen control-flow-induced discontinuities by interpolating between the outputs of all the relevant control-flow branches. Applying this approach to a particle simulation is a promising future research direction, but may come with large runtimes due to the very high number of control-flow branches encountered in a particle simulation.

As a well-known alternative or addition to pathwise derivatives, the *likelihood ratio* or *score function* method [34,35] proposes to compute a term

$$\mathbb{E}_\omega\left[\frac{\partial \log(p(\theta,\omega))}{\partial\theta}\cdot f(\theta,\omega)\right] = \mathbb{E}_\omega\left[\frac{\frac{\partial p}{\partial\theta}(\theta,\omega)}{p(\theta,\omega)}\cdot f(\theta,\omega)\right]. \tag{10}$$

This term accounts for the part of the derivative of $\mathbb{E}f$ related to a differentiable change of the probability $p(\theta,\omega)$ that discrete random events (e.g. whether an `if` or `else` branch is taken) turn out in the way they do when $f(\theta,\omega)$ is computed. As such, (10) should be added to $\mathbb{E}(f')$.

Indeed, for the function f_2 in Fig. 1, the bracketed expression in (10) evaluates to $[\frac{-d}{0.6}\cdot 1.0]$ when the `if` branch is taken (60 % probability) and to $[\frac{d}{0.4}\cdot 2.0]$ when the `else` branch is taken (40 % probability), giving an expected value of d. However, we were only able to determine the values of the probabilities $p = 0.6, 0.4$ in the denominators, and their derivatives $\frac{\partial p}{\partial\theta} = -d, d$ in the numerators, because the condition of the `if` statement is very simple.

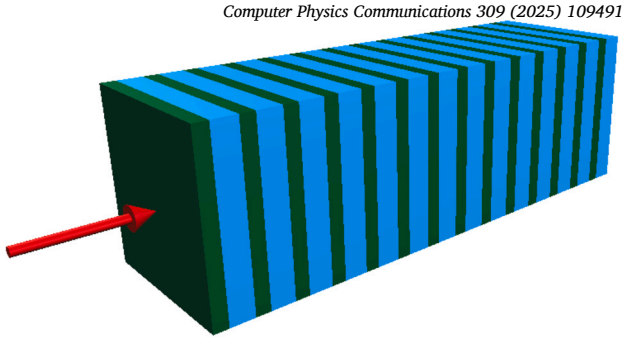

Fig. 2. Geometric structure of the sampling calorimeter. Figure courtesy of Novák et al. [26,27].

In the case of MC particle simulations, it is unclear how to determine p, because these simulations typically implement stochasticity by combining several random numbers and variables depending on the AD input θ, in non-linear ways. Additionally, the term (10) tends to have a high variance [36].

The *stochastic AD* method by Arya et al. [29] integrates certain kinds of discrete randomness into pathwise derivatives. For each intermediate value appearing in the MC program, this method keeps track of an alternative value that could have been attained with different random outcomes, and the derivative of the probability of such an outcome with respect to the AD input. While we consider it an interesting and promising approach, it appears not to be easily applicable to a MC particle simulation, because `if` statements and discrete randomness originating from comparisons of continuous random values are not yet supported.

Instead of trying to create an unbiased estimator for $(\mathbb{E}f)'$, in this work, we analyze the biased estimator $\overline{f'}$ for a MC code with full electromagnetic physics coverage but simple geometry. It turns out that when a single physics process called *multiple scattering* is disabled in our setup, the variance of $\overline{f'}$ is sufficiently low to obtain reliable estimates (7) of $\mathbb{E}(f')$ for moderate N_{diff}, and $\mathbb{E}(f')$ deviates from a difference quotient approximation of $(\mathbb{E}f)'$ only by a few percent. A bias of this magnitude can be perfectly acceptable as the derivatives only serve as a tool to guide optimization algorithms (and are not physical quantities that have to match measurements).

Section 2 gives an overview on the simulated hardware setup and the MC code, which we differentiated following the methodology described in Section 3. We then report on the stochastic noise of the MC code (Section 4.1), the variance and bias of the pathwise algorithmic derivative estimators (Section 4.2), and a simple demonstrator using these estimators for gradient-based optimization (Section 4.3), closing with conclusions and an outlook in Section 5.

2. Simulation of electromagnetic showers in a sampling calorimeter

2.1. Detector geometry

Fig. 2 shows the simple detector geometry used in this study. The detector hardware is a stack of n_l identical pairs of *absorber* and *gap layers*, each with a thickness of a and g (respectively) and transversal dimensions $d_t \times d_t$. The two types of layer are each made from homogeneous material; in particular, material properties are piecewise constant and change only at well-known two-dimensional volume boundaries. This assumption on the detector geometry is also made by Geant4 and is usually satisfied in practice. Primary particles arrive centered and orthogonally with an initial kinetic energy e.

A default set of values for these parameters is specified in Table 1. The setup was created by Novák et al. [26,27] and is based on Geant4's TestEm3 test case; however, the absorber material is lead tungstate ($PbWO_4$) instead of elementary lead (Pb), as a mixture of different atoms makes the test case more general. In this study, the primary energy e and

Table 1
Parameters of the simple sampling calorimeter geometry displayed in Fig. 2.

Parameter	Symbol	Arg.	Default value
Kinetic energy of primaries	e	`-e`	10 000 MeV
Thickness of absorber layers	a	`-a`	2.3 mm
Thickness of gap layers	g	`-g`	5.7 mm
Transversal dimension	d_t	`-t`	400 mm
Number of layers	n_l	`-l`	50
Type of primary particles		`-p`	electrons
Absorber material		$\binom{\text{JSON}}{\text{file}}$	$PbWO_4$
Gap material		$\binom{\text{JSON}}{\text{file}}$	liquid Ar

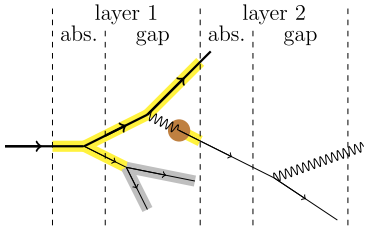

Fig. 3. Sketch of a very small shower consisting of electrons (lines) and photons (wiggly lines). Colors indicate different mechanisms leading to energy deposition in layer 1. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

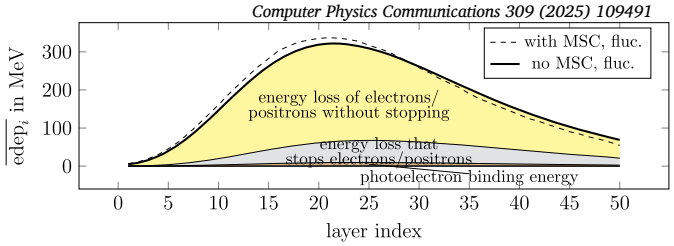

Fig. 4. HepEmShow-simulated average energy depositions in the 50 layers for $e = 10\,\text{GeV}$, $a = 2.3\,\text{mm}$, $g = 5.7\,\text{mm}$, with multiple scattering and energy loss fluctuations enabled (dashed line) or disabled (thick line) in the simulation. A breakdown of the energy deposition without MSC and fluctuations into the main energy deposition mechanisms is also shown (yellow, gray, brown).

Listing 1: Conceptual structure of a particle simulation.

```
for event in 1, ..., N:                                          # event loop
    create primary particle and push to stack
    while stack not empty:                                       # stacking loop
        pop particle from stack
        while particle not stopped:                              # stepping loop
            determine pathlength to next discrete physics process
                or hit of volume boundary
            move particle accordingly
            account for effects of physics processes (e. g. momentum
                change, energy deposition, secondary particles pushed
                onto the stack, stopping or annihilation of particle)
```

the layer thicknesses a and g will be considered as AD inputs θ, and all other parameters are considered constant.

2.2. Electromagnetic showers

Electrons, positrons and photons interact with the surrounding matter through various physical processes: ionization, bremsstrahlung, annihilation, pair production, the photoelectric effect, Compton scattering, etc. These processes happen at discrete points in time and can result in a loss of kinetic energy, change of momentum, deposition of energy in the surrounding matter, and/or creation of *secondary particles*. The interaction rates/*cross-sections* for these processes and their possible outcomes depend on the type and kinetic energy of the particle, and the material composition of the surrounding matter. Except for rare lepto- and photo-nuclear interactions that are neglected in the following, secondary particles are either electrons, positrons and photons. Secondary particles themselves interact with the surrounding matter, forming an electromagnetic shower.

A very small shower is sketched in Fig. 3. The brown circle indicates the emission of a photoelectron, depositing the K-shell binding energy of the ionized atom in the first gap layer. When electrons have lost all their kinetic energy, they stop and become part of the surrounding material (indicated by a gray background). Some of the aforementioned processes produce low-energetic particles very frequently; such interactions are usually modeled by a continuous energy loss along the entire path of the particle with a deterministic mean (yellow and gray background) and stochastic fluctuations. Very small but frequent changes of the momentum (*multiple scattering*, MSC) can be modeled by discrete changes of position and momentum, but this is not shown in Fig. 3.

The AD outputs $f(\theta, \omega)$ analyzed in this study are given by the energy depositions $\text{edep}_i(\theta, \omega)$ in the layers $i = 1, \ldots, 50$. Fig. 4 shows that disabling MSC (and energy loss fluctuations) has only a small effect on the energy depositions in our setup (as the dashed and thick lines are close to each other). The energy depositions without MSC and fluctuations are represented in Fig. 4 as a sum of the energy deposited by the continuous energy loss of electrons and positrons (sum of yellow and gray), and the much smaller binding energies that photoelectrons leave behind (brown); other energy deposition mechanisms are mostly irrelevant in our setup. The plotted data were obtained with a particle simulation, as detailed in the next section.

2.3. Particle simulations

Simulations of electromagnetic showers in material arrangements like the sampling calorimeter of section 2.1 can be thought of as a set of nested loops, as illustrated in Listing 1. Every iteration of the outermost *event loop* is concerned with a new primary particle, and contains a *stacking loop* that iterates over all particles in the resulting shower. Conceptually, each iteration of the innermost *stepping loop* determines the remaining pathlength until either a volume boundary is hit or a discrete physics process happens, and then moves the particle accordingly and accounts for any effects of physics processes. This is opposed to the particle simulator model studied by Kagan and Heinrich [31], which models material characteristics as a continuously changing three-dimensional field without explicit knowledge on the locations of volume boundaries.

The **Geant4** toolkit [1–3] covers a wide set of particles and processes, and has a very general way to handle geometry; accordingly, it is a very complex software project with around one million lines of code, mostly written in C++. The **G4HepEm** toolkit [25] isolates much of Geant4's models of physics processes in electromagnetic showers; e.g., G4HepEm's run-time functionality includes sampling of the distance to the next discrete interaction and sampling of interaction outcomes. On the one hand, G4HepEm can be used inside of Geant4 as an alternative to Geant4's native implementation of electromagnetic physics processes. Once the relevant material data (such as cross-sections) and other information have been pre-computed into a JSON file using separate initialization-time functionality of G4HepEm based on Geant4, G4HepEm's run-time functionality can also be used independently from Geant4, as a very compact standalone library for research and development activities in the field of HEP simulations. The **HepEmShow** package [26,27] consists of two applications: A *data generation* program using G4HepEm's initialization-time functionality and Geant4 to create the JSON file, and the main *simulation* that implements event, stacking and stepping loops in the sampling calorimeter setup described above (section 2.1), using physics information solely from G4HepEm's run-time functionality. HepEmShow's energy deposition results, represented

by the dashed line in Fig. 4, are in excellent agreement with Geant4-G4HepEm's [27].

2.4. Contributions and limitations

In this work, we differentiate the standalone run-time part of the G4HepEm toolkit and the HepEmShow simulation application. After disabling MSC in the simulation, we successfully validate our mean pathwise derivative estimator against difference quotients, observing only a small bias. To our knowledge, this is the first time that AD has been successfully applied to a full-fledged HEP simulation. Furthermore, we demonstrate the usefulness of these derivatives in a simple gradient-based optimization study.

While this is a major step on the way towards a differentiated Geant4-scale particle simulator, our setup makes the following key simplifications:

- As further detailed in section 4, we have to disable MSC in the simulation. Fig. 4 shows that this causes a minor change in the deposited energies in the simple sampling calorimeter setup considered by us, but it can potentially become more important for other use cases of Geant4.
- HepEmShow is made for one particular parametric geometry (Fig. 2) whereas Geant4 has a very general and flexible implementation of geometry.
- External electromagnetic fields, which exert forces on charged particles, are not available in HepEmShow but are fully supported in Geant4.
- HepEmShow is only meant for simulating electromagnetic showers consisting of electrons, positrons and photons, whereas Geant4 supports all particles relevant in HEP, and many models for physics processes across wide energy ranges.
- The present study is concerned with a limited set of AD inputs and outputs, whereas Geant4 users have very broad access to parameters and output data.

3. Pathwise algorithmic differentiation

The HepEmShow/G4HepEm simulation code computes the averaged per-layer energy depositions $\overline{\text{edep}_i}$ ($i = 1, \dots, 50$) from the input data in Table 1, notably the primary energy e, absorber thickness a and gap thickness g. We have applied AD to the simulation program in order to compute averaged per-layer derivatives

$$\frac{\partial\,\overline{\text{edep}_i}}{\partial\, e}, \frac{\partial\,\overline{\text{edep}_i}}{\partial\, a}, \frac{\partial\,\overline{\text{edep}_i}}{\partial\, g}.$$

To this end, we first applied the machine-code-based AD tool Derivgrind in a black-box fashion (Section 3.1). After first promising observations, we switched to the operator-overloading AD tool CoDiPack (Section 3.2) with a MC-specific tape size reduction technique to reduce memory usage in the reverse mode (Section 3.3).

3.1. Machine-code-based differentiation using derivgrind

Derivgrind [37] inserts AD logic into the machine code of the program to be differentiated. Therefore, only very little modification of the source code of G4HepEm and HepEmShow is required. Naturally, we had to change a few lines to indicate AD inputs (e, a, g) and outputs (edep_i) and to output the derivatives. In addition, a few G4HepEm-defined math functions like `G4Log` were replaced with their standard library counterparts (e.g. `std::log`), as their implementations perform real arithmetic via bit-wise manipulations of floating-point data in a way that might not be correctly understood by AD tools [21]. After exploratory experiments with Derivgrind's forward mode showed encouraging results, we decided to invest the time to apply an operator-overloading AD tool that offers much higher performance (see Table 2 for a comparison of run-times).

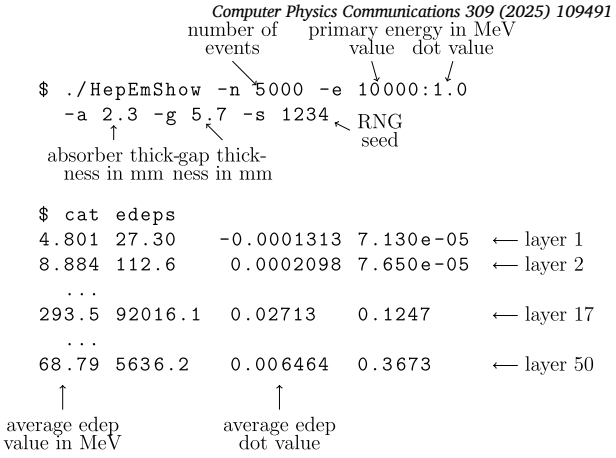

Fig. 5. User interface of the differentiated HepEmShow application in the forward mode. Dot values of inputs are specified in the command line interface (here, shown for `-e`). Dot values of outputs are written to a file `edeps`.

3.2. Operator-overloading differentiation using CoDiPack

Results presented in the remainder of this study were obtained by the operator-overloading AD tool CoDiPack [38]. In the shape of a C++ header, CoDiPack defines *AD types* that behave very similar to the built-in C++ floating-point types like `double`, but augment all real-arithmetic operations with AD logic. For maximal flexibility, we replaced most occurrences of `double` in the source codes of G4HepEm and HepEmShow with a type alias `G4double`, which we can set to `double`, `codi::RealForward` and `codi::RealReverse` to build non-AD, forward-mode and reverse-mode variants, respectively. The code is available at

https://github.com/SciCompKL/g4hepem/
https://github.com/SciCompKL/hepemshow/

No type exchange has been performed

- in the data generation part of HepEmShow producing the JSON file (containing pre-computed material data etc.), to avoid having to differentiate Geant4;
- in the JSON I/O library [39] used by the standalone part of G4HepEm – instead, conversions between `double`s in the library and `G4double`s in G4HepEm have been added to the interface; and
- for variables declared as `constexpr`, as they must have a *literal type* according to the C++ standards but the CoDiPack types are not literal.

In addition to the replacements of G4HepEm-defined math functions (Section 3.1), some manual refactoring of the source code was necessary around uses of the `?:`-operator and implicit casts to integers.

We have extended HepEmShow's I/O to allow the user to specify the AD inputs and outputs. As shown in Fig. 5, in the forward mode, the user can supply dot values of the primary energy e, absorber thickness a, and gap thickness g and access dot values of the average edeps. Reverse-mode HepEmShow requires an additional command-line argument `-b` with the adjoint values of the mean edeps in all layers, separated by colons, and output the adjoint values of e, a and g in a file.

For other variables used by HepEmShow, adding them as AD inputs and outputs would likely be straightforward. However, as we have not differentiated the initialization-time functionality (which uses Geant4), it is not possible at the moment to declare Geant4-internal data (e.g. cross-section tables) as AD inputs.

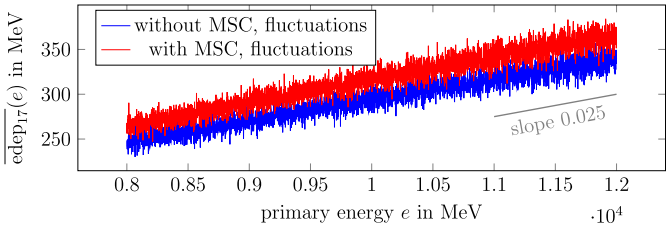

Fig. 6. Dependency of the simulated mean energy deposition in layer 17 on the primary energy e. For every point in this plot, $N = 100$ events were simulated using the same random seed. Fig. 7 zooms into this plot to see if these "noisy" functions are differentiable and if their derivatives match the large-scale slope of 0.025.

3.3. Reduction of the tape size

In the reverse mode of AD, operator-overloading AD tools record a *tape* data structure storing the real-arithmetic evaluation graph from the inputs to the outputs. For long-running programs, the tape size might exceed the amount of available memory. In our case, the recording of a single event loop iteration occupies roughly around 250 MB of memory on the tape (measured for $e = 10\,\text{GeV}$, $a = 2.3\,\text{mm}$, $g = 5.7\,\text{mm}$). As all event loop iterations run independently from each other, only a single iteration must be stored at a time, and the corresponding section of the tape can be evaluated and cleared at the end of each iteration to limit the tape size [40].

As source transformation tools have access to the entire source code of the function to be differentiated, they can generally use smaller tapes and apply more advanced code optimizations. As it is possible to compile the HepEmShow simulation and its G4HepEm dependency in a single translation unit, it would be worthwhile to investigate how compiler-based source transformation AD tools such as Clad [41] perform on the code base.

4. Results

4.1. Stochastic noise with and without multiple scattering

We first take a look at how the energy deposition depends on the primary particle energy e without AD, in order to be able to explain our findings with AD in the next section 4.2.

Large scale. Fig. 6 shows the simulated mean energy deposition in layer 17, $\overline{\text{edep}_{17}}$, averaged over $N = 100$ events per run, as a function of primary particle energy e. Each of the 4001 data points between $e = 8000\,\text{MeV}$ and $12\,000\,\text{MeV}$ was produced by a separate run of HepEmShow, always using the same initial random seed. The experiment has been conducted with the full set of electromagnetic processes available in G4HepEm (red), and with a simplified setup that had MSC and energy loss fluctuations disabled (blue).

The number of $N = 100$ simulated events for Fig. 6 is very small, so the standard deviation of the mean (3) is rather large, causing the clearly visible stochastic noise. This is expected: If e is perturbed even very slightly, the control flow in the simulator is likely to change at some point, making a different number of calls to the RNG and thus leaving it in a different state for the subsequent execution, which is therefore entirely uncorrelated even though the same RNG seed has been used initially [42]. Choosing a higher N reduces the amplitude of the stochastic noise, but does not eliminate it.

Despite the noise, Fig. 6 shows a clear large-scale trend, with $\overline{\text{edep}_{17}}(e)$ rising, in both setups, approximately linearly by 100 MeV over the entire range of e spanning 4000 MeV. Thus, the derivative $(\mathbb{E}\,\text{edep}_{17})'$ of the expected energy deposition at $e = 10\,000\,\text{MeV}$ can be estimated as

$$(\mathbb{E}\,\text{edep}_{17})' := \frac{\partial}{\partial e}\left[\mathbb{E}_\omega \text{edep}_{17}(e,\omega)\right] \approx \frac{100\,\text{MeV}}{4000\,\text{MeV}} = 0.025. \tag{11}$$

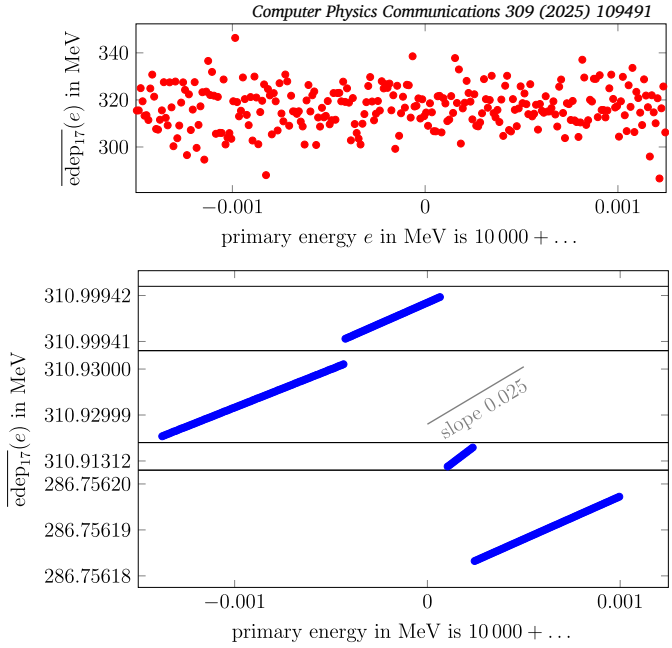

Fig. 7. Zoom into Fig. 6, showing a much smaller range of e. Again, each point represents a HepEmShow simulation of $N = 100$ events, always using the same random seed. The energy deposition computed with the full set of physics processes still appears noisy (top). With multiple scattering and energy loss fluctuations disabled, however, the averaged energy deposition is a piecewise differentiable function of the primary energy, and its derivative (i.e. the slope of the segments) approximately matches the large-scale slope.

This *large-scale* slope is what is relevant for e.g. optimization purposes, so this is what we want to compute. For validation purposes, we approximate the large-scale derivative using difference quotients similar to the right-hand side of (11), taking care that a sufficiently large number of events is simulated as difference quotients are poorly conditioned.

When we apply AD to a code computing $\overline{\text{edep}_{17}}$, we obtain the floating-point accurate, *local* slope $\overline{\text{edep}'_{17}}$ of the algorithm implemented in the code. To read this local slope from the plot, we have to zoom in.

Small scale. Fig. 7 shows $\overline{\text{edep}_{17}}(e)$ plotted over a much more narrow interval, again using the same seed for all runs of HepEmShow. For the full physics setup, we observe the same noisy behavior (top figure), even if we zoomed in further. With MSC and energy loss fluctuations turned off, however, the function is clearly piecewise differentiable (bottom figure). This qualitative difference is very important for AD, as it allows us to confirm that the slopes of the differentiable segments (which is what pathwise AD computes) are close to the large-scale slope of about 0.025 as determined in (11) which we want to compute. There is still more than one jump per keV on the horizontal axis, due to discrete randomness and decorrelating RNG states as mentioned above. These jumps are much larger in magnitude than the differentiable evolution in between, and they are responsible for the noise visible in Fig. 6. However, the differentiable evolution in between the jumps already accounts for (approximately) the entire large-scale evolution.

In fact, it is not necessary to disable energy loss fluctuations; a plot in the style of Fig. 7, with only multiple scattering turned off, has more discontinuities but still clearly visible increasing differentiable segments.

Summary. The qualitative analysis conducted in this section for a single layer indicates that after disabling MSC, there is only a small difference between

- the large-scale derivative $(\mathbb{E}\,\text{edep}_{17})'$ required for applications and approximated by difference quotients, and
- the local derivative $\overline{\text{edep}'_{17}}$, computed by AD without special care for randomness (thus treating random numbers as constants), which

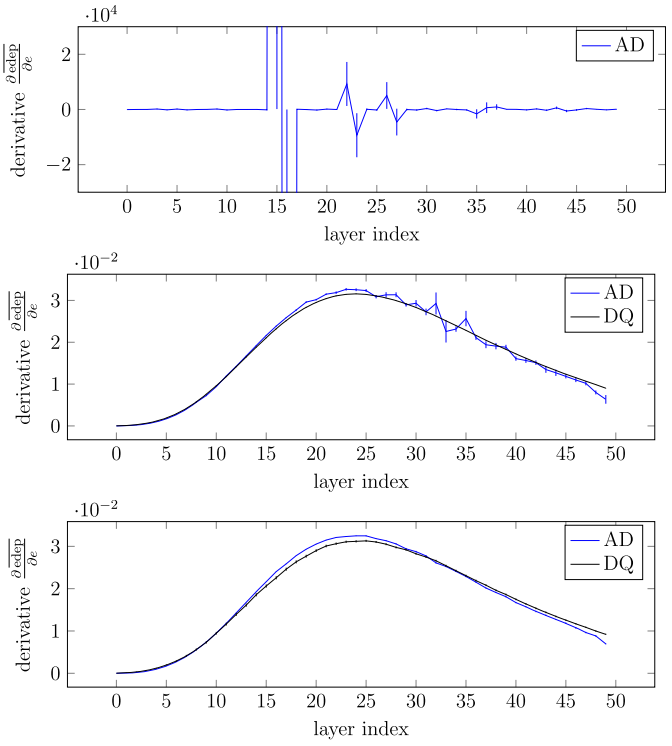

Fig. 8. Algorithmic derivative of the mean edep in the calorimeter layers with respect to the primary energy e (blue), and the corresponding difference quotients (black). Error bars indicate 68 %-confidence intervals (i.e. plus/minus one standard deviation). Top: Default configuration of G4HepEm with all physics processes, means over 24 M events. Middle: All physics processes except for multiple scattering, means over 24 M events. Bottom: 864 M simulated events to reduce the stochastic error, and a smaller interval for the difference quotient to reduce the truncation error.

approximates $\mathbb{E}(\mathrm{edep}'_{17})$ in the limit of many simulated events ($N_{\mathrm{diff}} \to \infty$) by the strong law of large numbers.

In the next section 4.2, we study this hypothesis quantitatively and in more generality, looking at algorithmic derivatives of energy depositions in all layers with respect to e, a and g.

4.2. Variance and bias of pathwise algorithmic derivatives

In this section, we collect results obtained with our CoDiPack-differentiated version of HepEmShow/G4HepEm (sections 3.2, 3.3).

Pathwise derivatives of the full simulation code including MSC are noisy. Fig. 8 shows the mean pathwise forward-mode algorithmic derivative of the simulated energy deposition $\overline{\mathrm{edep}_i(e)}$ in all the calorimeter layers $i = 0, \ldots, 49$, with respect to the initial kinetic energy e of the primary particles, at $e = 10\,\mathrm{GeV}$. For the top plot, 24 M events were simulated using the full list of physics processes. Mean pathwise derivatives $\overline{\mathrm{edep}'_i(e)}$ of the code seem to have a very large variance and deviate by orders of magnitudes from the value of 0.025 suggested by (11). Averaging over many more events might reduce noise, but as the number of events would need to rise by a factor of 10^{12} to bring a standard deviation of the order of 10^4 down to the order of 10^{-2}, this is not feasible in practice.

It should be noted that this observation does not imply that the physical phenomenon of MSC itself would be inherently non-differentiable. We can only infer that G4HepEm's algorithm implementing the Urban MSC model [43] has noisy algorithmic derivatives. This could be related to the often-heard statement that "black-box" differentiation of iterative numerical algorithms may compute wrong derivatives [44] and knowledge on the mathematical structure behind them should thus be included into the AD implementation. For the remainder of this study, however, we disable MSC in the simulation, and leave the development of an AD-friendly MSC model to further research.

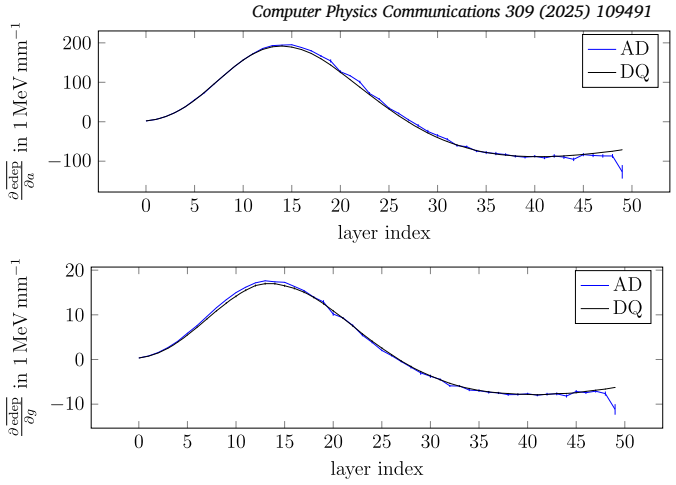

Fig. 9. Algorithmic derivative of the edep with respect to the absorber thickness a (top) and gap thickness g (bottom).

Disabling MSC leads to low variance and bias for pathwise derivatives. The middle plot of Fig. 8 shows the averaged result of 24 M AD runs at $e = 10\,\mathrm{GeV}$ with MSC disabled in the simulation. Additionally, 24 M primal runs without MSC at $e = 9.9\,\mathrm{GeV}$ and $e = 10.1\,\mathrm{GeV}$ were conducted to compute a central difference quotient (DQ) that approximates the large-scale slope $(\mathbb{E}\,\mathrm{edep}_i(e))'$. Both plots match very well. Thus, disabling multiple scattering is the key algorithmic change that allows us to obtain algorithmic derivatives with a sufficiently low variance and an expected value close to the numerical derivatives. Error bars in Fig. 8 indicate plus/minus one standard deviation of the derivative approximation; it should be noted that the error bars of the difference quotient approximation are very tight because it is evaluated on a rather large interval ($\left|\frac{\pm 0.1\,\mathrm{GeV}}{10\,\mathrm{GeV}}\right| = 1\%$), and that they do not include any numerical truncation error.

The bias with respect to difference quotients is around 5%. The bottom plot of Fig. 8 has been created with 864 M samples to decrease the stochastic error, and a more narrow interval 9.995…10.005 GeV for the difference quotient to decrease the numerical truncation error, again with MSC disabled. We observe a statistically highly significant but low deviation of the mean pathwise derivative approximating $\mathbb{E}(\mathrm{edep}'_i)$ from the difference quotients approximating $(\mathbb{E}\,\mathrm{edep}_i)'$. Except for the first and last few layers, the relative error of the derivatives is around 5 %.

Similar observations can be made for derivatives w. r. t. layer thicknesses. Fig. 9 shows that algorithmic derivatives of the energy deposition with respect to the absorber and gap thicknesses as well have a sufficiently low variance and bias (w. r. t. difference quotients) when MSC is turned off.

Algorithmic and numeric derivatives deviate much more for individual edep mechanisms. While the mean pathwise derivatives of the total energy depositions edep_i are close to the large-scale derivatives approximated by difference quotients, as described above, this does not hold on the level of individual mechanisms to register energy deposition in the simulation code.

We have used Fig. 4 to illustrate that most of the energy deposition comes from continuous energy loss, followed by the binding energy of photoelectrons. In fact, G4HepEm registers continuous energy loss at two main places in the code: As a side action next to another physical process or a change of volumes in the geometry (indicated in yellow) and as the sole action if it uses up all the remaining kinetic energy of the particle (indicated in gray). Fig. 10 shows the derivatives of these three terms, with respect to the primary energy e again. Interestingly, algorithmic and numeric derivatives do not match.

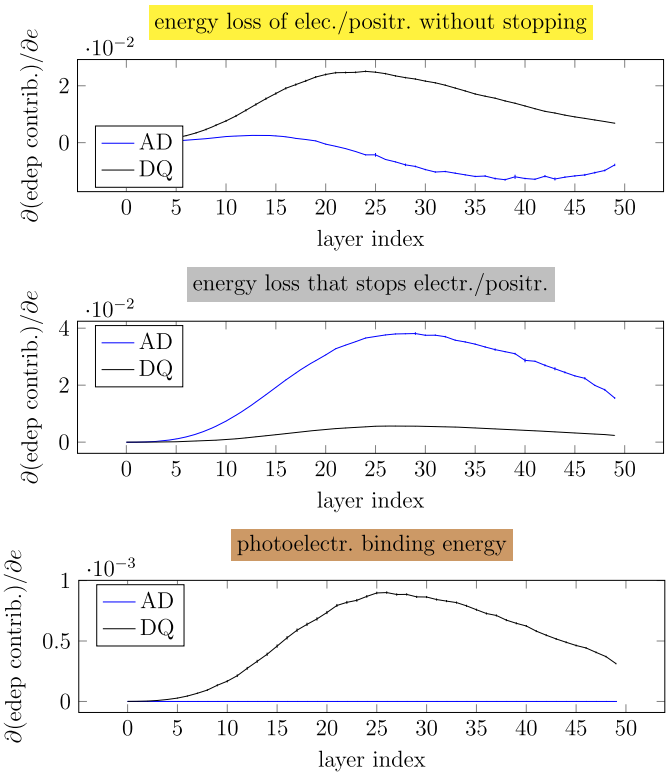

Fig. 10. Breakdown of $\partial \,\mathrm{edep}_i / \partial\, e$ into the dominating energy deposition mechanisms.

To understand the algorithmic derivatives of the two energy loss contributions, let us imagine that the incoming electron in Fig. 3 had an infinitesimally higher initial kinetic energy. This would allow the primary and the secondary electrons to travel infinitesimally further before they stop, making the gray segments longer and their energy deposition higher. Thus, this mechanism contributes most to the algorithmic derivative of the energy deposition.

Regarding the difference quotients, we have to imagine a small but non-infinitesimal increase in the initial kinetic energy. As before, gray segments become longer, but one of the secondary electrons may now have enough energy to reach the next layer, and the energy loss would become a side action. Therefore, difference quotients mainly see an increase in energy loss that does not stop the particle.

We should note that the distinction between the two mechanisms to register continuous energy loss comes from modeling and coding considerations and is not rooted in physics. Our observation that algorithmic and numerical derivatives deviate heavily for the two individual mechanisms, even though they approximately match for the sum, shows that care should be taken to only declare physically meaningful data as AD outputs.

Concerning the deposition of binding energies in photoelectric effect events, difference quotients register an increase that could be caused by more events taking place, and/or an increasing probability of elements with higher binding energies to be selected as the ionized atom from the material. Both types of dependencies have the structure of f_2 in Fig. 1, and are thus not seen by pathwise algorithmic derivatives, which are therefore zero. This illustrates that we cannot expect pathwise derivatives to perfectly match the numerical derivatives. Note that the order of magnitude of the missing photoelectric binding energy part of $\frac{\partial\, \overline{\mathrm{edep}}}{\partial\, e}$ (around 10^{-3}, as displayed in the bottom plot of Fig. 10), is similar to the error of $\frac{\partial \overline{\mathrm{edep}}}{e}$ itself (displayed in Fig. 8). However, the binding energy is not the only source of error, as the latter is positive for some layers and negative for other layers.

Table 2
Runtime (in seconds) and memory (in MB) required to simulate 10 000 electron events for $e = 5, 10, 20$ GeV. In comparison, the run-times of forward-mode AD using the exploratory AD tool Derivgrind (section 3.1) for 100 electron events are 61 s / 114 s / 218 s, corresponding to a slow-down of the primal simulation by a factor of around 70.

Primary	5 GeV		10 GeV		20 GeV	
energy	time	mem.	time	mem.	time	mem.
primal	84	5.7	163	5.6	320	5.7
forward mode	147 (×1.8)	5.9	287 (×1.8)	5.9	558 (×1.7)	5.9
reverse mode	452 (×5.4)	111	867 (×5.3)	195	1662 (×5.2)	284

Performance Measurements. Table 2 shows the runtime and memory consumption of a HepEmShow simulation of 10 000 electrons, in terms of user time and maximum resident set size measured on an exclusive 2.6 GHz Intel Xeon Gold 6126 node at the Elwetritsch cluster of the University of Kaiserslautern-Landau.

Forward-mode and reverse-mode AD using CoDiPack slow down the program by factors of around 1.8 and 5.4, respectively.

Memory consumption increased slightly in the forward mode because CoDiPack's forward-mode type has twice the size of a `double`. In the reverse mode, the tape occupies a significant but perfectly manageable amount of memory, which grows with the primary energy.

4.3. Optimization using averages of pathwise derivatives

This section deals with an application of pathwise algorithmic derivatives for gradient-based optimization.

The *gradient descent algorithm* attempts to find the minimizer $\theta^* \in \mathbb{R}^n$ of a loss function $L : \mathbb{R}^n \to \mathbb{R}$, starting from an initial guess $\theta^{(0)} \in \mathbb{R}^n$, by iteratively computing better and better "candidate minimizers" $\theta^{(1)}$, $\theta^{(2)}, \ldots$ via

$$\theta_j^{(k+1)} = \theta_j^{(k)} - d_j^{(k)} \cdot \frac{\partial L}{\partial \theta_j}(\theta^{(k)}). \tag{12}$$

The factors $d_j^{(k)}$ are called *step-sizes* or *learning rates*, and may be fixed or computed adaptively. When stochastic estimates are used instead of the actual gradient, the scheme is known as *stochastic gradient descent* (SGD). In machine learning, stochastic estimates of loss function gradients typically result from computing the loss on a randomly selected subset of the training data instead of the entire data. In our case, outputs of the MC simulation, and hence their derivatives, are stochastic already by definition. Deviations of the estimated derivatives from the true values steer the optimizer into a less ideal direction, but it can still arrive at the minimum, maybe with a larger number of steps.

Automated Design of Scientific Instruments. To demonstrate that the stochastic and biased pathwise AD gradient estimator can indeed be useful for optimization, we have designed the following simple parameter identification problem. The parameters in Table 1 have been used to simulate a target edep distribution $\overline{\mathrm{edep}}_i$ across the layers $i = 0, \ldots, 49$ of the calorimeter, shown in Fig. 4. From this target edep distribution, we wish to infer $e^* = 10\,\mathrm{GeV}$ and $a^* = 2.3\,\mathrm{mm}$, assuming that we only know the other parameters in Table 1. The task of identifying a primary energy value e^* that leads to a prescribed energy deposition curve is a model problem for applications where the position of physical interactions should be controlled, e.g. for experiment design or in radiation therapy planning. Derivatives with respect to a geometric parameter a^* could be useful for detector optimization problems.

To identify e^* and a^*, we have to search for the minimizer of the loss function L given by the squared error of the resulting edep distribution,

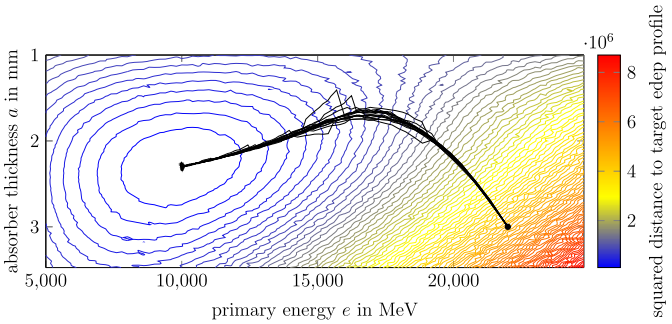

Fig. 11. Reconstruction of the values of primary energy and absorber thickness that lead to a given energy deposition profile in a sampling calorimeter, using the gradient descent optimizer with algorithmic derivatives of the shower simulation.

$$L(e,a) = \sum_{i=0}^{49} \left(\overline{\mathrm{edep}_i}(e,a) - \overline{\mathrm{edep}_i}(e^*,a^*) \right)^2 . \tag{13}$$

Fig. 11 shows 16 paths of the stochastic gradient descent scheme across the loss landscape of L. We have chosen a step-size of 1 for e and $10^{-7}\,\mathrm{mm}^2\,\mathrm{MeV}^{-2}$ for a to account for their different units and orders of magnitude, and estimate the gradient using 1 k events in each step, for 350 steps. Starting from $e^{(0)} = 22\,\mathrm{GeV}$ and $a^{(0)} = 3\,\mathrm{mm}$, the SGD optimizer robustly converges to the minimizer (e^*, a^*). There is room for further investigation of optimal choices of such hyperparameters; e.g., the optimization succeeds even with only 100 events per step.

5. Conclusion and outlook

Conclusion. In this work, we have successfully applied AD to a Monte-Carlo simulation of electromagnetic showers in a sampling calorimeter, in order to compute pathwise derivatives of the energy depositions with respect to the energy of the primary particles and the thicknesses of the layers. The simulation models all the relevant physics processes, while the detector geometry has been kept rather simple. Applying AD to the code without any algorithmic changes led to algorithmic derivatives of very high variance, but the only problem seems to be that black-box pathwise AD is not the right tool to differentiate the algorithm used to model multiple scattering in G4HepEm. With multiple scattering disabled, variances of algorithmic derivatives are sufficiently low and their means are close to the (numerical) derivatives of the average energy depositions, with a deviation of about 5 %. Errors of this magnitude may be perfectly acceptable when the derivatives are used for gradient-based optimization, as demonstrated by a simple parameter identification study.

Outlook. In order to scale our encouraging result to the full generality of Geant4, we propose the following next steps:

- It could be worthwhile to apply a high-performance AD tool to the Geant4 codebase, in order to try to reproduce the findings of our present work with Geant4's G4HepEm physics process. This would allow to see if Geant4's very general implementation of geometry is an obstacle for AD, and if not, allow to consider many different detector layouts. Subsequently, Geant4's G4HepEm physics process could be replaced by the native Geant4 electromagnetic physics processes. We have reported encouraging preliminary results in a preprint [45].
- Additional efforts should be dedicated to analyze and mitigate the incompatibility of AD with multiple scattering, potentially by creating an AD-friendly MSC model.
- One could then enable uniform and non-uniform electromagnetic fields in the simulation to check their compatibility with AD.
- To conclude physics generalizations, it would be interesting to include other particles and, in particular, enable hadronic processes.
- At some point, it may become necessary to go beyond mean pathwise derivatives, and employ and improve differentiable and probabilistic programming tooling to account for discrete randomness; e.g. following the approaches of StochasticAD [29] or DiscoGrad [33].
- In particular, systematic efforts should be dedicated to source transformation AD tools to enable the above workflows.
- Once any pre- and postprocessing software is differentiated as well, algorithmic derivatives can be used to efficiently optimize actual experiment designs in their planning phase.

CRediT authorship contribution statement

Max Aehle: Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Mihály Novák:** Writing – review & editing, Supervision, Software, Methodology, Conceptualization. **Vassil Vassilev:** Writing – review & editing, Supervision, Project administration, Conceptualization. **Nicolas R. Gauger:** Writing – review & editing, Supervision, Resources, Funding acquisition, Conceptualization. **Lukas Heinrich:** Writing – review & editing, Conceptualization. **Michael Kagan:** Writing – review & editing, Conceptualization. **David Lange:** Writing – review & editing, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to thank members of the MODE collaboration, in particular Tommaso Dorigo, for valuable discussions that helped pave the way to this research project.

MA and NG gratefully acknowledge the funding of the research training group SIVERT by the German Federal State of Rhineland-Palatinate. Also, MA and NG gratefully acknowledge the funding of the German National High Performance Computing (NHR) Association for the Center NHR South-West. MK is supported by the US Department of Energy (DOE) under grant DE-AC02-76SF00515. DL and VV are supported by the National Science Foundation under Grant OAC-2311471. LH is supported by the Excellence Cluster ORIGINS, which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2094-390783311.

This research was supported by the Munich Institute for Astro-, Particle and BioPhysics (MIAPbP), which is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2094-390783311. Computing resources have been provided by the Alliance for High Performance Computing in Rhineland-Palatinate (AHRP) via the Elwetritsch cluster at the University of Kaiserslautern-Landau.

Data availability

Data will be made available on request.

References

[1] S. Agostinelli, et al., Geant4—a simulation toolkit, Nucl. Instrum. Methods Phys. Res., Sect. A, Accel. Spectrom. Detect. Assoc. Equip. 506 (3) (2003) 250–303, https://doi.org/10.1016/S0168-9002(03)01368-8, http://www.sciencedirect.com/science/article/pii/S0168900203013688.

[2] J. Allison, et al., Geant4 developments and applications, IEEE Trans. Nucl. Sci. 53 (1) (2006) 270–278, https://doi.org/10.1109/TNS.2006.869826, http://ieeexplore.ieee.org/document/1610988/.

[3] J. Allison, et al., Recent developments in Geant4, Nucl. Instrum. Methods Phys. Res., Sect. A, Accel. Spectrom. Detect. Assoc. Equip. 835 (2016) 186–225, https://doi.org/10.1016/j.nima.2016.06.125, https://linkinghub.elsevier.com/retrieve/pii/S0168900216306957.

[4] T.A. Albring, M. Sagebaum, N.R. Gauger, Efficient aerodynamic design using the discrete adjoint method in su2, in: 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, American Institute of Aeronautics and Astronautics, Inc., Washington, D.C., 2016, https://arc.aiaa.org/doi/abs/10.2514/6.2016-3518.

[5] D.F. Shanno, Conditioning of quasi-Newton methods for function minimization, Math. Comput. 24 (111) (1970) 647–656, https://doi.org/10.1090/S0025-5718-1970-0274029-X, https://www.ams.org/mcom/1970-24-111/S0025-5718-1970-0274029-X/.

[6] A. Griewank, A. Walther, Evaluating Derivatives, Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, PA, US, 2008, https://epubs.siam.org/doi/book/10.1137/1.9780898717761.

[7] H. Fischer, Automatic differentiation of the vector that solves a parametric linear system, J. Comput. Appl. Math. 35 (1) (1991) 169–184, https://doi.org/10.1016/0377-0427(91)90205-X, https://www.sciencedirect.com/science/article/pii/037704279190205X.

[8] H. Xie, J.-G. Liu, L. Wang, Automatic differentiation of dominant eigensolver and its applications in quantum physics, Phys. Rev. B 101 (2020) 245139, https://doi.org/10.1103/PhysRevB.101.245139, https://link.aps.org/doi/10.1103/PhysRevB.101.245139.

[9] P. Glasserman, Estimating Sensitivities, Springer New York, New York, NY, 2003, pp. 377–420.

[10] T.-M. Li, M. Aittala, F. Durand, J. Lehtinen, Differentiable Monte Carlo ray tracing through edge sampling, ACM Trans. Graph. (Proc. SIGGRAPH Asia) 37 (6) (2018) 222:1–222:11.

[11] S. Bangaru, T.-M. Li, F. Durand, Unbiased warped-area sampling for differentiable rendering, ACM Trans. Graph. 39 (6) (2020) 245:1–245:18.

[12] T. Zeltner, S. Speierer, I. Georgiev, W. Jakob, Monte Carlo estimators for differential light transport, in: Transactions on Graphics (Proceedings of SIGGRAPH), vol. 40(4), 2021.

[13] S. Bangaru, J. Michel, K. Mu, G. Bernstein, T.-M. Li, J. Ragan-Kelley, Systematically differentiating parametric discontinuities, ACM Trans. Graph. 40 (107) (2021) 107:1–107:17.

[14] J. Michel, K. Mu, X. Yang, S.P. Bangaru, E.R. Collins, G. Bernstein, J. Ragan-Kelley, M. Carbin, T.-M. Li, Distributions for compositionally differentiating parametric discontinuities, Proc. ACM Program. Lang. 8 (OOPSLA1) (Apr. 2024), https://doi.org/10.1145/3649843.

[15] R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Proceedings of the 12th International Conference on Neural Information Processing Systems, NIPS'99, MIT Press, Cambridge, MA, USA, 1999, pp. 1057–1063.

[16] C. Tang, R.R. Salakhutdinov, Learning stochastic feedforward neural networks, in: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 26, Curran Associates, Inc., 2013, https://proceedings.neurips.cc/paper_files/paper/2013/file/d81f9c1be2e08964bf9f24b15f0e4900-Paper.pdf.

[17] D.P. Kingma, M. Welling, Auto-encoding variational bayes, in: Y. Bengio, Y. LeCun (Eds.), 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014, http://arxiv.org/abs/1312.6114.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 27, Curran Associates, Inc., 2014, https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[19] S. Mohamed, M. Rosca, M. Figurnov, A. Mnih, Monte Carlo gradient estimation in machine learning, J. Mach. Learn. Res. 21 (132) (2020) 1–62, http://jmlr.org/papers/v21/19-346.html.

[20] T. Dorigo, A. Giammanco, P. Vischia, M. Aehle, M. Bawaj, A. Boldyrev, P. de Castro Manzano, D. Derkach, J. Donini, A. Edelen, F. Fanzago, N.R. Gauger, C. Glaser, A.G. Baydin, L. Heinrich, R. Keidel, J. Kieseler, C. Krause, M. Lagrange, M. Lamparth, L. Layer, G. Maier, F. Nardi, H.E. Pettersen, A. Ramos, F. Ratnikov, D. Röhrich, R.R. de Austri, P.M.R. del Árbol, O. Savchenko, N. Simpson, G.C. Strong, A. Taliercio, M. Tosi, A. Ustyuzhanin, H. Zaraket, Toward the end-to-end optimization of particle physics instruments with differentiable programming, Rev. Phys. 10 (2023) 100085, https://doi.org/10.1016/j.revip.2023.100085, https://www.sciencedirect.com/science/article/pii/S2405428323000047.

[21] M. Aehle, L. Arsini, R.B. Barreiro, A. Belias, F. Bury, S. Cebrian, A. Demin, J. Dickinson, J. Donini, T. Dorigo, M. Doro, N.R. Gauger, A. Giammanco, L. Gray, B.S. González, V. Kain, J. Kieseler, L. Kusch, M. Liwicki, G. Maier, F. Nardi, F. Ratnikov, R. Roussel, R.R. de Austri, F. Sandin, M. Schenk, B. Scarpa, P. Silva, G.C. Strong, P. Vischia, Progress in end-to-end optimization of detectors for fundamental physics with differentiable programming, arXiv:2310.05673, 2023.

[22] P. de Castro, T. Dorigo, INFERNO: inference-aware neural optimisation, Comput. Phys. Commun. 244 (2019) 170–179, https://doi.org/10.1016/j.cpc.2019.06.007, https://www.sciencedirect.com/science/article/pii/S0010465519301948.

[23] N. Simpson, L. Heinrich, Neos: end-to-end-optimised summary statistics for high energy physics, J. Phys. Conf. Ser. 2438 (1) (2023) 012105, https://doi.org/10.1088/1742-6596/2438/1/012105, https://dx.doi.org/10.1088/1742-6596/2438/1/012105.

[24] A.G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, M. Ma, X. Zhao, P. Torr, V. Lee, K. Cranmer Prabhat, F. Wood, Etalumis: bringing probabilistic programming to scientific simulators at scale, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '19, Association for Computing Machinery, New York, NY, USA, 2019.

[25] M. Novák, J. Hahnfeld, et al., G4HepEm, https://github.com/mnovak42/g4hepem, 2020.

[26] M. Novák, The HepEmShow R&D Project, https://github.com/mnovak42/hepemshow, 2023.

[27] M. Novák, HepEmShow: a compact EM shower simulation application, https://hepemshow.readthedocs.io/en/latest/, 2023.

[28] R. Durrett, Probability: theory and examples, version 5 January 11, 2019, https://services.math.duke.edu/~rtd/PTE/PTE5_011119.pdf, 2019.

[29] G. Arya, M. Schauer, F. Schäfer, C. Rackauckas, Automatic differentiation of programs with discrete randomness, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems, vol. 35, Curran Associates, Inc., 2022, pp. 10435–10447, https://proceedings.neurips.cc/paper_files/paper/2022/file/43d8e5fc816c692f342493331d5e98fc-Paper-Conference.pdf.

[30] A.K. Lew, M. Huot, S. Staton, V.K. Mansinghka, Adev: sound automatic differentiation of expected values of probabilistic programs, Proc. ACM Program. Lang. 7 (POPL) (2023) 121–153, https://doi.org/10.1145/3571198.

[31] M. Kagan, L. Heinrich, Branches of a tree: taking derivatives of programs with discrete and branching randomness in high energy physics, arXiv:2308.16680, 2023.

[32] S. Christodoulou, U. Naumann, Differentiable programming: efficient smoothing of control-flow-induced discontinuities, arXiv:2305.06692, https://arxiv.org/abs/2305.06692, 2023.

[33] J.N. Kreikemeyer, P. Andelfinger, Smoothing methods for automatic differentiation across conditional branches, IEEE Access 11 (2023) 143190–143211, https://doi.org/10.1109/access.2023.3342136.

[34] P.W. Glynn, Likelihood ratio gradient estimation for stochastic systems, Commun. ACM 33 (10) (1990) 75–84, https://doi.org/10.1145/84537.84552.

[35] J. Schulman, N. Heess, T. Weber, P. Abbeel, Gradient estimation using stochastic computation graphs, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, NIPS'15, MIT Press, Cambridge, MA, USA, 2015, pp. 3528–3536.

[36] D.J. Rezende, S. Mohamed, D. Wierstra, Stochastic backpropagation and approximate inference in deep generative models, in: E.P. Xing, T. Jebara (Eds.), Proceedings of the 31st International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 32, PMLR, Bejing, China, 2014, pp. 1278–1286.

[37] M. Aehle, J. Blühdorn, M. Sagebaum, N.R. Gauger, Forward-mode automatic differentiation of compiled programs, arXiv:2209.01895 [cs], Sep. 2022, http://arxiv.org/abs/2209.01895.

[38] M. Sagebaum, T. Albring, N.R. Gauger, High-performance derivative computations using CoDiPack, ACM Trans. Math. Softw. 45 (4) (Dec. 2019), https://doi.org/10.1145/3356900.

[39] N. Lohmann, JSON for modern C++, https://github.com/nlohmann/json, 2024.

[40] L. Hascoët, S. Fidanova, C. Held, Adjoining independent computations, in: G. Corliss, C. Faure, A. Griewank, L. Hascoët, U. Naumann (Eds.), Automatic Differentiation of Algorithms: From Simulation to Optimization, Springer, New York, New York, NY, 2002, pp. 299–304.

[41] V. Vassilev, M. Vassilev, A. Penev, L. Moneta, V. Ilieva, Clad – Automatic Differentiation Using Clang and LLVM, vol. 608, IOP Publishing, 2015, p. 012055, https://iopscience.iop.org/article/10.1088/1742-6596/608/1/012055/pdf.

[42] M. Aehle, J. Alme, G.G. Barnaföldi, J. Blühdorn, T. Bodova, V. Borshchov, A. van den Brink, V. Eikeland, G. Feofilov, C. Garth, N.R. Gauger, O. Grøttvik, H. Helstrup, S. Igolkin, R. Keidel, C. Kobdaj, T. Kortus, L. Kusch, V. Leonhardt, S. Mehendale, R.N. Mulawade, O.H. Odland, G. O'Neill, G. Papp, T. Peitzmann, H.E.S. Pettersen, P. Piersimoni, R. Pochampalli, M. Protsenko, M. Rauch, A.U. Rehman, M. Richter, D. Röhrich, M. Sagebaum, J. Santana, A. Schilling, J. Seco, A. Songmoolnak, Ákos Sudár, G. Tambave, I. Tymchuk, K. Ullaland, M. Varga-Kofarago, L. Volz, B. Wagner, S. Wendzel, A. Wiebel, R. Xiao, S. Yang, S. Zillien, Exploration of differentiability in a proton computed tomography simulation framework, Phys. Med. Biol. 68 (24) (2023) 244002, https://doi.org/10.1088/1361-6560/ad0bdd.

[43] L. Urban, A model for multiple scattering in Geant4, Dec. 2006.

[44] J.C. Gilbert, Automatic differentiation and iterative processes, Optim. Methods Softw. 1 (1) (1992) 13–21, https://doi.org/10.1080/10556789208805503.

[45] M. Aehle, X.T. Nguyen, M. Novák, T. Dorigo, N.R. Gauger, J. Kieseler, M. Klute, V. Vassilev, Efficient forward-mode algorithmic derivatives of geant4, arXiv:2407.02966, https://arxiv.org/abs/2407.02966, 2024.