Differentiating GATE/Geant4 with Derivgrind

Max Aehle   Nicolas R. Gauger

Scientific Computing Group,
University of Kaiserslautern-Landau

& SIVERT Research Training Group

Third MODE Workshop on
Differentiable Programming for Experiment Design

Princeton University, July 24th–26th, 2023

Background

Goal: Optimize the design of particle physics instruments with e. g. gradient-based optimization methods.

Objective Function: $J(\text{design parameters } x) = (\text{physics performance}) - \mu \cdot (\text{cost})$

Gradient descent to minimize J: $x_{\text{better}} = x_{\text{old}} - \alpha \cdot \nabla J(x_{\text{old}})$

How to find ∇J? Algorithmic Differentiation / Differentiable Programming.[1]

[1] Set of techniques to evaluate derivatives of computer-implemented functions.

[2] Monte-Carlo simulator of the passage of particles through matter. 850k lines of C++. Toolkits like GATE might add substantial amounts of code in C++ or other languages.

Background

Goal: Optimize the design of particle physics instruments with e. g. gradient-based optimization methods.

Objective Function: $J(\text{design parameters } x) = (\text{physics performance}) - \mu \cdot (\text{cost})$

Gradient descent to minimize J: $x_{\text{better}} = x_{\text{old}} - \alpha \cdot \nabla J(x_{\text{old}})$

How to find ∇J? Algorithmic Differentiation / Differentiable Programming.[1]

If the detector design x affects particle trajectories, simulators like Geant4[2] might appear in J. What to do?

- Apply AD/DP directly to Geant4.
  - ⊖ technically difficult
  - ⊖ many discontinuities
  - ⊕ most general approach

[1] Set of techniques to evaluate derivatives of computer-implemented functions.

[2] Monte-Carlo simulator of the passage of particles through matter. 850k lines of C++. Toolkits like GATE might add substantial amounts of code in C++ or other languages.

Background

Goal: Optimize the design of particle physics instruments with e. g. gradient-based optimization methods.

Objective Function: $J(\text{design parameters } x) = (\text{physics performance}) - \mu \cdot (\text{cost})$

Gradient descent to minimize J: $x_{\text{better}} = x_{\text{old}} - \alpha \cdot \nabla J(x_{\text{old}})$

How to find ∇J? Algorithmic Differentiation / Differentiable Programming.[1]

If the detector design x affects particle trajectories, simulators like Geant4[2] might appear in J. What to do?

- Neglect this dependency.
- Apply AD/DP directly to Geant4.
  - ⊖ technically difficult
  - ⊖ many discontinuities
  - ⊕ most general approach

[1] Set of techniques to evaluate derivatives of computer-implemented functions.

[2] Monte-Carlo simulator of the passage of particles through matter. 850k lines of C++. Toolkits like GATE might add substantial amounts of code in C++ or other languages.

Background

Goal: Optimize the design of particle physics instruments with e. g. gradient-based optimization methods.

Objective Function: $J(\text{design parameters } x) = (\text{physics performance}) - \mu \cdot (\text{cost})$

Gradient descent to minimize J: $x_{\text{better}} = x_{\text{old}} - \alpha \cdot \nabla J(x_{\text{old}})$

How to find ∇J? Algorithmic Differentiation / Differentiable Programming.[1]

If the detector design x affects particle trajectories, simulators like Geant4[2] might appear in J. What to do?

- Neglect this dependency.
- Use surrogate models.
- Apply AD/DP directly to Geant4.
  - ⊖ technically difficult
  - ⊖ many discontinuities
  - ⊕ most general approach

[1] Set of techniques to evaluate derivatives of computer-implemented functions.

[2] Monte-Carlo simulator of the passage of particles through matter. 850k lines of C++. Toolkits like GATE might add substantial amounts of code in C++ or other languages.

Background

Goal: Optimize the design of particle physics instruments with e. g. gradient-based optimization methods.

Objective Function: $J(\text{design parameters } x) = (\text{physics performance}) - \mu \cdot (\text{cost})$

Gradient descent to minimize J: $x_{\text{better}} = x_{\text{old}} - \alpha \cdot \nabla J(x_{\text{old}})$

How to find ∇J? Algorithmic Differentiation / Differentiable Programming.[1]

If the detector design x affects particle trajectories, simulators like Geant4[2] might appear in J. What to do?

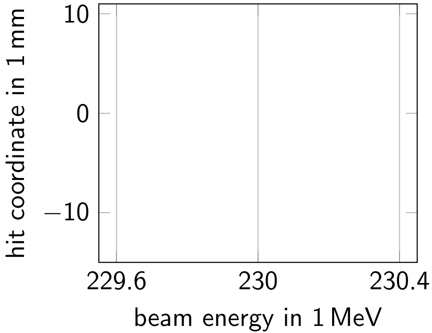
- Neglect this dependency.
- Use surrogate models.
- Apply AD/DP directly to Geant4. ⊖ technically difficult ⊖ many discontinuities ⊕ most general approach

[1] Set of techniques to evaluate derivatives of computer-implemented functions.

[2] Monte-Carlo simulator of the passage of particles through matter. 850k lines of C++. Toolkits like GATE might add substantial amounts of code in C++ or other languages.

GATE/Geant4 Setup

- GATE is a medical imaging toolkit built on top of Geant4.
- In our setup, a single energetic proton passes through a human head and a digital tracking calorimeter (DTC) of the Bergen pCT collaboration.

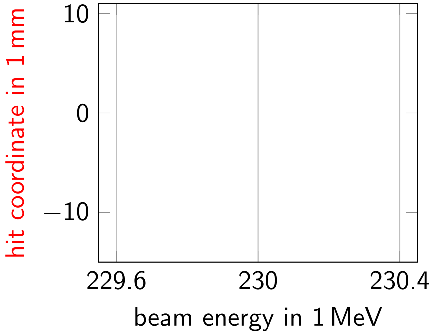


- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

GATE/Geant4 Setup

- GATE is a medical imaging toolkit built on top of Geant4.
- In our setup, a single energetic proton passes through a human head and a digital tracking calorimeter (DTC) of the Bergen pCT collaboration.
- AD outputs ("derivative of..."): x-coordinates of hits in the first two tracking layers of the DTC.

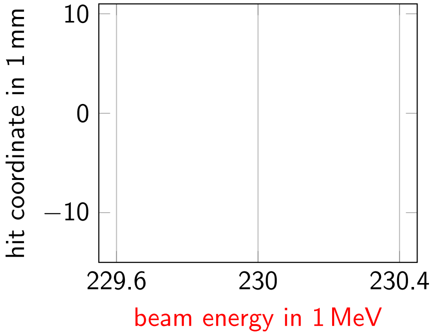


- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

GATE/Geant4 Setup

- GATE is a medical imaging toolkit built on top of Geant4.
- In our setup, a single energetic proton passes through a human head and a digital tracking calorimeter (DTC) of the Bergen pCT collaboration.
- AD outputs ("derivative of..."): *x*-coordinates of hits in the first two tracking layers of the DTC.
- AD input ("with respect to ..."): beam energy.

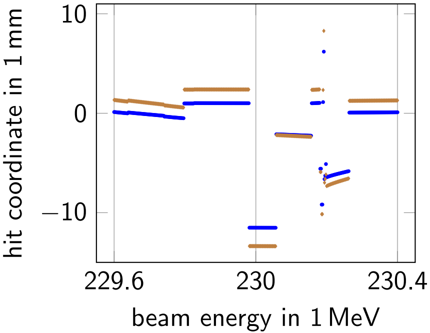


- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

GATE/Geant4 Setup

- GATE is a medical imaging toolkit built on top of Geant4.
- In our setup, a single energetic proton passes through a human head and a digital tracking calorimeter (DTC) of the Bergen pCT collaboration.
- AD outputs ("derivative of..."): x-coordinates of hits in the first two tracking layers of the DTC.
- AD input ("with respect to ..."): beam energy.
- The seed of the random number generator was fixed.



- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

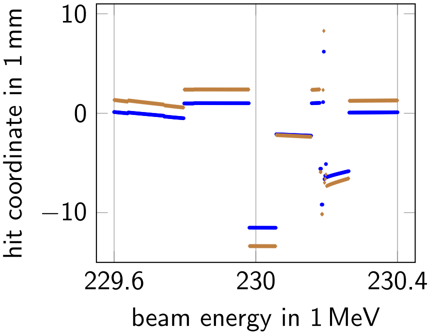
GATE/Geant4 Setup

Why are there jumps?
At some point, a different physics process is selected. It consumes a different amount of random numbers, so the subsequent execution receives a shifted (i. e. entirely different) sequence of random numbers.

Is the function differentiable between the jumps?
It looks so, let's check at

$$x_0 = 230\,\text{MeV}$$

by evaluating difference quotients.



- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

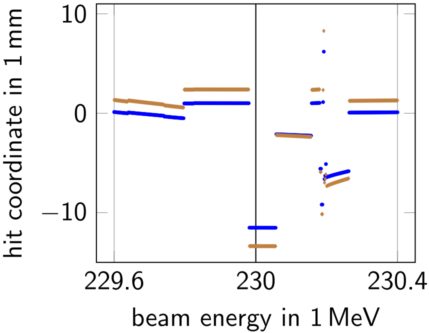
GATE/Geant4 Setup

Why are there jumps?
At some point, a different physics process is selected. It consumes a different amount of random numbers, so the subsequent execution receives a shifted (i. e. entirely different) sequence of random numbers.

Is the function differentiable between the jumps?
It looks so, let's check at

$$x_0 = 230\,\text{MeV}$$

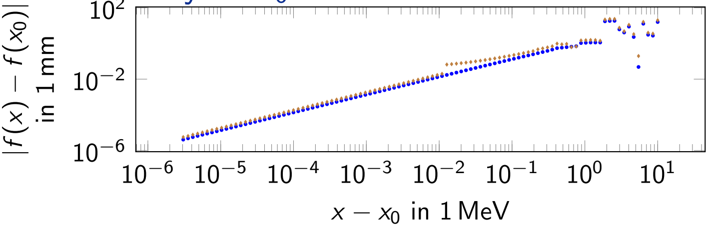
by evaluating difference quotients.



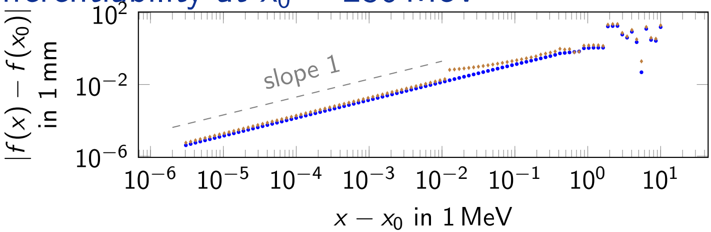
- First tracking layer.
- Second tracking layer.

Adapted from Aehle, Alme et al., Derivatives in Proton CT.

Differentiability at $x_0 = 230\,\text{MeV}$



Differentiability at $x_0 = 230\,\text{MeV}$



⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\text{mm}}{\text{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653

Algorithmic Differentiation with Derivgrind

- AD tools allow to compute the derivatives of computer-implemented functions ("primal programs").
- To this end, they need to find out about the real-arithmetic computations performed by the primal program.
- Most AD tools do this in the source code or as part of the compiler.
- Derivgrind operates on machine code just before it runs on the CPU, achieving an unprecedented degree of independence from the source code of the primal program.

Derivgrind is available at `https://github.com/SciCompKL/derivgrind`

Declaring Inputs and Outputs to Derivgrind

```
+ #include "/somepath/include/valgrind/derivgrind.h"
```

“Seeding” dot values of the input variable (beam energy):

```
   if (command == pIonCmd) {
     pSourcePencilBeam->SetIonParameter(newValue);
   }
   if (command == pEnergyCmd) {
     double energy = pEnergyCmd->GetNewDoubleValue(newValue);
+    double one = 1.0;
+    DG_SET_DOTVALUE(&energy,&one,sizeof(double));
     pSourcePencilBeam->SetEnergy(energy);
   }
```

Obtaining dot values of output variables (hit position):

```
   if (m_rootHitFlag) m_treeHit->Fill();
+  float pos = *(float*)(m_treeHit->GetBranch("posX")->GetAddress());
+  float pos_d;
+  DG_GET_DOTVALUE(&pos,&pos_d,sizeof(float));
+  std::cout << "pos_d=" << pos_d << "\n";
```

Running GATE/Geant4 under Derivgrind

```
valgrind --tool=derivgrind Gate ⟨args⟩
```

Running GATE/Geant4 under Derivgrind

`valgrind --tool=derivgrind Gate ⟨args⟩`

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.

Running GATE/Geant4 under Derivgrind

```
valgrind --tool=derivgrind Gate ⟨args⟩
```

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:

Running GATE/Geant4 under Derivgrind

```
valgrind --tool=derivgrind Gate ⟨args⟩
```

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.

Running GATE/Geant4 under Derivgrind

```
valgrind --tool=derivgrind Gate ⟨args⟩
```

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.
  - Constants have dot value 0, the AD input x has dot value 1.

Running GATE/Geant4 under Derivgrind

`valgrind --tool=derivgrind Gate` ⟨*args*⟩

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.
  - Constants have dot value 0, the AD input x has dot value 1.
  - When data is copied around, copy dot value alongside.

Running GATE/Geant4 under Derivgrind

```
valgrind --tool=derivgrind Gate ⟨args⟩
```

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.
  - Constants have dot value 0, the AD input x has dot value 1.
  - When data is copied around, copy dot value alongside.
  - When there is a real-arithmetic operation like $c = b \cdot a$, use differentiation rule like $\dot{c} = \dot{a} \cdot b + a \cdot \dot{b}$.

Running GATE/Geant4 under Derivgrind

`valgrind --tool=derivgrind Gate` ⟨*args*⟩

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.
  - Constants have dot value 0, the AD input x has dot value 1.
  - When data is copied around, copy dot value alongside.
  - When there is a real-arithmetic operation like $c = b \cdot a$, use differentiation rule like $\dot{c} = \dot{a} \cdot b + a \cdot \dot{b}$.
- The Valgrind framework runs the instrumented VEX IR on a synthetic CPU.

Running GATE/Geant4 under Derivgrind

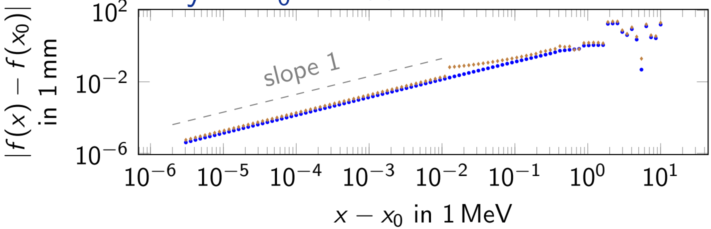
```
valgrind --tool=derivgrind Gate ⟨args⟩
```

What happens now?

- The Valgrind framework loads the `Gate` executable and all libraries and translates them into the VEX intermediate representation.
- Our Derivgrind tool adds AD logic into the VEX IR, e. g. in the forward mode with single input x:
  - Keep track of *dot value* $\dot{a} = \frac{\partial a}{\partial x}$ for every value a.
  - Constants have dot value 0, the AD input x has dot value 1.
  - When data is copied around, copy dot value alongside.
  - When there is a real-arithmetic operation like $c = b \cdot a$, use differentiation rule like $\dot{c} = \dot{a} \cdot b + a \cdot \dot{b}$.
- The Valgrind framework runs the instrumented VEX IR on a synthetic CPU.

$\rightsquigarrow$ We see the original output, plus some Valgrind/Derivgrind messages, plus the dot values of the output variables.

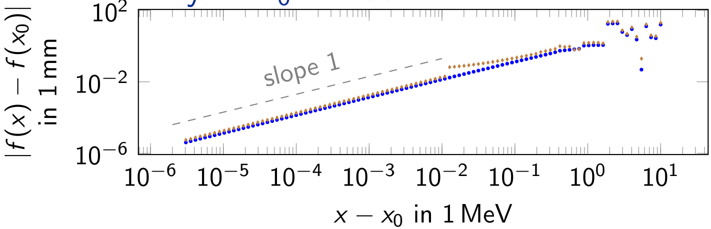
Differentiability at $x_0 = 230\,\mathrm{MeV}$



$\rightsquigarrow$ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\mathrm{mm}}{\mathrm{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653

Differentiability at $x_0 = 230\,\mathrm{MeV}$



⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\mathrm{mm}}{\mathrm{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653
Derivgrind, forward mode	−0.068512	−0.113841

Bit-Tricks in Geant4

- Geant4 defines and uses a function `G4Log` adapted from the VDT math library.
- `G4Log` first performs a “range reduction”, scaling its argument by power of 2 to map it into $[0.5, 1)$.
- This is done by setting the exponent bits to `0b01111111110` via bitwise operations.

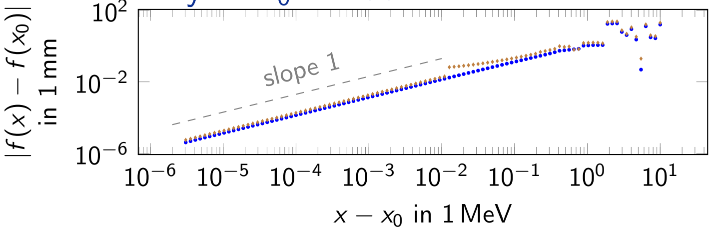
Bit-Tricks in Geant4

- Geant4 defines and uses a function `G4Log` adapted from the VDT math library.
- `G4Log` first performs a "range reduction", scaling its argument by power of 2 to map it into $[0.5, 1)$.
- This is done by setting the exponent bits to `0b01111111110` via bitwise operations.
- Derivgrind's dot value propagation does not recognize this as an arithmetic operation.

Bit-Tricks in Geant4

- Geant4 defines and uses a function `G4Log` adapted from the VDT math library.
- `G4Log` first performs a "range reduction", scaling its argument by power of 2 to map it into $[0.5, 1)$.
- This is done by setting the exponent bits to `0b01111111110` via bitwise operations.
- Derivgrind's dot value propagation does not recognize this as an arithmetic operation.
- Fix: Edit Geant4 source code, replacing body of `G4Log` by call to `log`.

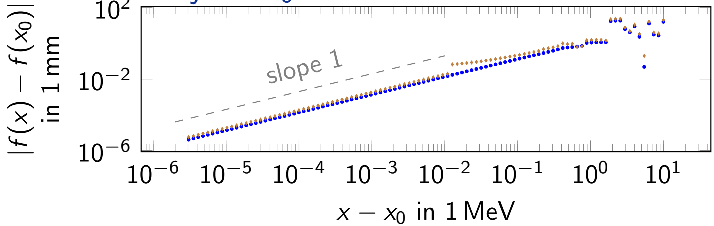
Differentiability at $x_0 = 230\,\text{MeV}$



⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\text{mm}}{\text{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653
Derivgrind, forward mode	−0.068512	−0.113841

Differentiability at $x_0 = 230\,\mathrm{MeV}$



⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\mathrm{mm}}{\mathrm{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653
Derivgrind, forward mode	−0.068512	−0.113841
... after replacing `G4Log`	−0.081339	−0.130524

⇝ After the fix, Derivgrind's derivative agrees well with difference quotients (everything compiled in debug mode).

Declaring Inputs and Outputs, Reverse Mode

```
+ #include "/somepath/include/valgrind/derivgrind.h"
```

Declaring input variable (beam energy):

```
    if (command == pIonCmd) {
      pSourcePencilBeam->SetIonParameter(newValue);
    }
    if (command == pEnergyCmd) {
      double energy = pEnergyCmd->GetNewDoubleValue(newValue);
+     DG_INPUTF(energy);
      pSourcePencilBeam->SetEnergy(energy);
    }
```

Declaring output variables (hit position):

```
    if (m_rootHitFlag) m_treeHit->Fill();
+   float pos = *(float*)(m_treeHit->GetBranch("posX")->GetAddress());
+   DG_OUTPUTF(pos);
```

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value a.

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value a.
  - Constants have index 0, AD inputs are assigned a “fresh” index.

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value *a*.
  - Constants have index 0, AD inputs are assigned a "fresh" index.
  - When data is copied around, copy indices alongside.

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value *a*.
  - Constants have index 0, AD inputs are assigned a “fresh” index.
  - When data is copied around, copy indices alongside.
  - When there is a real-arithmetic operation, record the indices and partial derivatives of the operands and assign a new index to the result.

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value *a*.
  - Constants have index 0, AD inputs are assigned a “fresh” index.
  - When data is copied around, copy indices alongside.
  - When there is a real-arithmetic operation, record the indices and partial derivatives of the operands and assign a new index to the result.

⇝ This produces a file `dg-tape` with the real-arithmetic evaluation tree.

Running GATE/Geant4 under Derivgrind, Reverse Mode

```
valgrind --tool=derivgrind --record=$PWD Gate ⟨args⟩
```

What happens now?

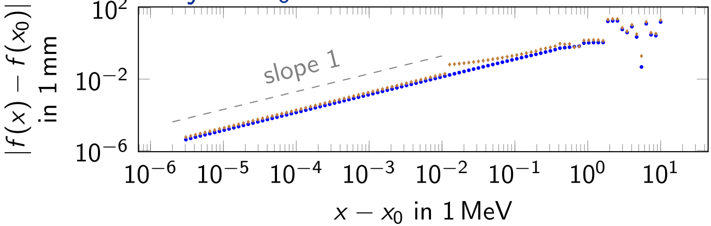
- The same procedure as in the forward mode, but with recording-pass AD logic:
  - Keep track of an *index* for every value a.
  - Constants have index 0, AD inputs are assigned a “fresh” index.
  - When data is copied around, copy indices alongside.
  - When there is a real-arithmetic operation, record the indices and partial derivatives of the operands and assign a new index to the result.

⇝ This produces a file `dg-tape` with the real-arithmetic evaluation tree.

```
tape-evaluation $PWD
```

⇝ Computes the derivatives.

Differentiability at $x_0 = 230\,\text{MeV}$

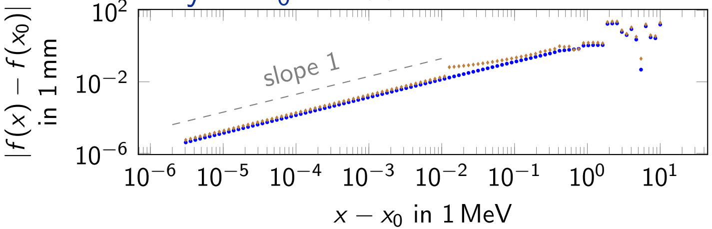


⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\text{mm}}{\text{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653
Derivgrind, forward mode	−0.068512	−0.113841
... after replacing `G4Log`	−0.081339	−0.130524

⇝ After the fix, Derivgrind's derivative agrees well with difference quotients (everything compiled in debug mode).

Differentiability at $x_0 = 230\,\mathrm{MeV}$



⇝ The hit coordinate $f(x)$ is differentiable in the beam energy x at x_0.

Derivatives in $\frac{\mathrm{mm}}{\mathrm{MeV}}$	• first layer	♦ second layer
Difference quotient	−0.082016	−0.130653
Derivgrind, forward mode	−0.068512	−0.113841
... after replacing `G4Log`	−0.081339	−0.130524
... in reverse mode	−0.081339	−0.130524

⇝ After the fix, Derivgrind's derivative agrees well with difference quotients (everything compiled in debug mode).

Summary

- Derivgrind implements forward-mode AD and operator-overloading-style reverse-mode AD for compiled programs like GATE/Geant4.
- Very limited access to primal source code needed, only in order to identify input/output variables (and for debugging in case of failure).
- In the case of unsupported bit-tricks, Derivgrind might miss or misunderstand real-arithmetic dependencies.
- In our GATE/Geant4 setup, this happened but could be fixed (in debug mode).

Next Steps

- Create bit-trick-finding tool. Right now we can discover the `G4Log` bit-trick automatically, but we don't know yet why Derivgrind fails for the release-mode GATE/Geant4.
- Work on more difficult example and connect AD derivatives to optimizer.
  - Can anyone propose a simple yet meaningful Geant4 setup with some inputs, some outputs, and an objective function defined on them?
- Try out other AD tools, e. g.
  - CoDiPack – operator-overloading tool developed in Kaiserslautern.
  - Enzyme or CLAD – compiler-based; Geant4 libs can be built statically.

Contact Information

Max Aehle, max.aehle@scicomp.uni-kl.de

Nicolas R. Gauger, nicolas.gauger@scicomp.uni-kl.de

Chair for Scientific Computing
University of Kaiserslautern-Landau (RPTU)

Derivgrind is available at https://github.com/SciCompKL/derivgrind

Project E-Mail Address: derivgrind@projects.rptu.de

Video (7 min) about Derivgrind+LibreOffice Calc:
https://t1p.de/tt4ne

supported by TU NACHWUCHSRING