Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

The STCE Scripting Language

Support for Algorithmic Differentiation

Uwe Naumann

Informatik 12 (STCE), RWTH Aachen University

Contents

Outline

- dco::gt1s<T>::type enables the propagation of tangents (directional derivatives) by overloading of all arithmetic operations and intrinsic functions in C++.
- dco::ga1s<T>::type uses overloading to generate a *tape* that records data dependences and elemental partial derivatives. Adjoints are propagated by interpretation of the tape.
- Recursive nesting of tangent and adjoint types yields higher-order tangents and adjoints.
- Conceptually, the entire range of functionalities provided by dco/c++ is available for STCE scripting; see

  `www.nag.com/content/algorithmic-differentiation-software/`.

Outline

Tangent Algorithmic Differentiation (AD) of differentiable programs implementing multivariate vector functions

$$f : \boldsymbol{R}^n \to \boldsymbol{R}^m : y = f(x)$$

yields the tangent program

$$f^{(1)} : \boldsymbol{R}^n \times \boldsymbol{R}^n \to \boldsymbol{R}^m \times \boldsymbol{R}^m : \ (y, y^{(1)}) = f^{(1)}(x, x^{(1)})$$

defined as

$$f^{(1)} \equiv \left(f(x), f'(x) \cdot x^{(1)} \right) \ .$$

The Jacobian $f' \equiv \frac{df}{dx} \in \boldsymbol{R}^{m \times n}$ can be computed with a relative (with respect to the cost a evaluating f) cost of $\mathcal{O}(n)$ by letting $x^{(1)}$ range over the Cartesian basis vectors in $\boldsymbol{R}^n$.

Tangents with dco/c++


```
1  #include<iostream>
2  using namespace std;
3
4  #include "dco.hpp" // dco/c++ header
5
6  int main() {
7    dco::gt1s<float>::type x,y; // tangent type variables
8    dco::value(x)=2; // set primal value of active input
9    dco::derivative(x)=1; // seed tangent of active input
10   y=sin(x); // evaluate primal and propagate tangent
11   cout << dco::value(y) << " "; // get primal value of active output
12   cout << dco::derivative(y) << endl; // harvest tangent of active output
13   return 0;
14 }
```

yields output

```
0.909297
−0.416147
```

Eigen and dco/c++ can be combined to yield derivatives of linear algebra programs.

For example, differentiation of the solution of the system of linear equations

$$A \cdot x = b \,,$$

where $A \in \boldsymbol{R}^{n \times n}$ is invertible and $x, b \in \boldsymbol{R}^n$, with respect to b yields

$$\frac{dx}{db} = \frac{d(A^{-1} \cdot b)}{db} = A^{-1}$$

which is easily validated by the following sample code.

Tangents and Eigen


```
1  #include<iostream>
2  using namespace std;
3
4  #include "Eigen.hpp"
5  #include "dco.hpp"
6
7  int main() { // computes the inverse of A as dx/db
8    const int n=3;
9    using T=dco::gt1s<float>::type; // tangent type
10   using VT=Eigen::vector_t<T>; // vector of tangent type variables
11   using MT=Eigen::matrix_t<T>; // matrix of tangent type variables
12   VT x(n), b=VT::Random(n); MT A=MT::Random(n,n);
13   for (int i=0;i<n;i++) {
14     dco::derivative(b(i))=1; // seed
15     x=A.lu().solve(b); // propagate
16     for (int j=0;j<n;j++) cout << dco::derivative(x(j)) << " "; // harvest
17     cout << endl;
18     dco::derivative(b(i))=0; // unseed
19   }
20   cout << A.inverse().transpose() << endl;
21   return 0;
22 }
```

Outline

Adjoint AD of differentiable programs implementing multivariate vector functions

$$f : \boldsymbol{R}^n \to \boldsymbol{R}^m : \; y = f(x)$$

yields the adjoint program

$$f_{(1)} : \boldsymbol{R}^n \times \boldsymbol{R}^m \to \boldsymbol{R}^m \times \boldsymbol{R}^n : \; (y, x_{(1)}) = f_{(1)}(x, y_{(1)})$$

defined as

$$f_{(1)} \equiv \left(f(x), f'(x)^T \cdot y_{(1)} \right) \; .$$

The Jacobian can be computed with a relative cost of $\mathcal{O}(m)$ by letting $y_{(1)}$ range over the Cartesian basis vectors in $\boldsymbol{R}^m$.

Potentially, the size of the *tape* to be recorded for the necessary reversal of the data flow could exceed the available memory resources. Techniques beyond black-box adjoint AD need to be considered in this case.

Adjoints with dco/c++


```
1  #include<iostream>
2  using namespace std;
3
4  #include "dco.hpp" // dco/c++ header
5
6  int main() {
7    dco::ga1s<float>::type x,y; // adjoint type variables
8    dco::value(x)=2; // set primal value of active input
9    dco::smart_tape_ptr_t<dco::ga1s<float>> tape; // enable recording of tape
10   tape->register_variable(x); // register active input with tape (should be read-only)
11   y=sin(x); // evaluate primal and record tape
12   tape->register_output_variable(y); // register active output (to makes it write-only)
13   cout << dco::value(y) << endl; // get primal value of active input
14   dco::derivative(y)=1; // seed adjoint of active output
15   tape->interpret_adjoint(); // propagate adjoint
16   cout << dco::derivative(x) << endl; // harvest adjoint of active input
17   return 0;
18 }
```

yields the same output as the tangent.

Adjoints and Eigen


```
1  #include<iostream>
2  using namespace std;
3
4  #include "Eigen.hpp"
5  #include "dco.hpp"
6
7  int main() { // computes the inverse of A a dx/db
8    const int n=3;
9    using M=dco::ga1s<float>; using T=M::type;
10   using VT=Eigen::vector_t<T>; using MT=Eigen::matrix_t<T>;
11   VT x(n), b=VT::Random(n); MT A=MT::Random(n,n);
12   dco::smart_tape_ptr_t<M> tape; // enable recording of tape
13   for (int i=0;i<n;i++) tape->register_variable(b(i)); // register active input
14   x=A.lu().solve(b); // record tape
15   for (int i=0;i<n;i++) tape->register_output_variable(x(i)); // register active output
16   for (int i=0;i<n;i++) {
17     dco::derivative(x(i))=1; // seed
18     tape->interpret_adjoint(); // propagate adjoint
19     for (int j=0;j<n;j++) cout << dco::derivative(b(j)) << " "; // harvest
20     tape->zero_adjoints(); // prepare for re-seed
21   }
22   return 0;
23 }
```

Outline

Second-Order Tangents

Tangent AD of a differentiable tangent program

$$f^{(1)} : \boldsymbol{R}^n \times \boldsymbol{R}^n \to \boldsymbol{R}^m \times \boldsymbol{R}^m : \; f^{(1)}(x, x^{(1)}) =: (y, y^{(1)})$$

yields the second-order tangent program

$$\begin{aligned} f^{(1,2)} &: \boldsymbol{R}^n \times \boldsymbol{R}^n \times \boldsymbol{R}^n \times \boldsymbol{R}^n \to \boldsymbol{R}^m \times \boldsymbol{R}^m \times \boldsymbol{R}^m \times \boldsymbol{R}^m \\ &: f^{(1,2)}(x, x^{(2)}, x^{(1)}, x^{(1,2)}) =: (y, y^{(2)}, y^{(1)}, y^{(1,2)}) \end{aligned}$$

defined using index notation for the Hessian tensor product (summation over common index) as

$$f^{(1,2)} \equiv \left(f(x), f'(x) \cdot x^{(2)}, f'(x) \cdot x^{(1)}, f''(x)_{k,j,i} \cdot x_j^{(1)} \cdot x_i^{(2)} + f'(x) \cdot x^{(1,2)} \right) .$$

The Hessian $f'' \equiv \frac{d^2 f}{dx^2} \in \boldsymbol{R}^{m \times n \times n}$ can be computed with a relative cost of $\mathcal{O}(n^2)$ by letting $x^{(1)}$ (first first-order tangent) and $x^{(2)}$ (second first-order tangent) range independently over the Cartesian basis vectors in $\boldsymbol{R}^n$ while setting $x^{(1,2)} := 0$ (second-order tangent).

Second-Order Tangents with dco/c++


```
1  #include<iostream>
2  using namespace std;
3
4  #include "dco.hpp" // dco/c++ header
5
6  int main() {
7    dco::gt1s<dco::gt1s<float>::type>::type x,y; // second-order tangent type variables
8    dco::value(dco::value(x))=2; // set primal value of active input
9    dco::value(dco::derivative(x))=1; // seed first first-order tangent
10   dco::derivative(dco::value(x))=1; // seed second first-order tangent
11   // second-order tangent value of active input is equal to zero by construction
12   y=sin(x); // propagate (primal and tangents)
13   cout << dco::value(dco::value(y)) << " "; // get primal value of active output
14   cout << dco::value(dco::derivative(y)) << " "; // harvest first first-order tangent
15   cout << dco::derivative(dco::value(y)) << " "; // harvest second first-order tangent
16   cout << dco::derivative(dco::derivative(y)) << endl; // harvest second-order tangent
17   return 0;
18 }
```

yields the output 0.909297 −0.416147 −0.416147 −0.909297 similar to ...

Outline

Second-Order Adjoints

Tangent AD of a differentiable adjoint program[1]

$$f_{(1)} : \boldsymbol{R}^n \times \boldsymbol{R}^m \to \boldsymbol{R}^m \times \boldsymbol{R}^n : \; f_{(1)}(x, y_{(1)}) =: (y, x_{(1)})$$

yields the second-order adjoint program

$$\begin{aligned} f^{(2)}_{(1)} &: \boldsymbol{R}^n \times \boldsymbol{R}^n \times \boldsymbol{R}^m \times \boldsymbol{R}^m \to \boldsymbol{R}^m \times \boldsymbol{R}^m \times \boldsymbol{R}^n \times \boldsymbol{R}^n \\ &: f^{(2)}_{(1)}(x, x^{(2)}, y_{(1)}, y^{(2)}_{(1)}) =: (y, y^{(2)}, x_{(1)}, x^{(2)}_{(1)}) \end{aligned}$$

defined as

$$f^{(2)}_{(1)} \equiv \left(f(x), f'(x) \cdot x^{(2)}, f'(x)^T \cdot y_{(1)}, f''(x)_{k,j,i} \cdot y_{(1)_k} \cdot x^{(2)}_i + f'(x)^T \cdot y^{(2)}_{(1)} \right) .$$

The Hessian can be computed with a relative cost of $\mathcal{O}(m \cdot n)$ by letting $y_{(1)}$ (first-order adjoint) and $x^{(2)}$ (first-order tangent) range independently over the Cartesian basis vectors in $\boldsymbol{R}^m$ and $\boldsymbol{R}^n$, respectively, while setting $y^{(2)}_{(1)} := 0$ (second-order adjoint).

[1] The remaining two combinations work too.

Second-Order Adjoints with dco/c++


```
1  #include<iostream>
2  using namespace std;
3
4  #include "dco.hpp" // dco/c++ header
5
6  int main() {
7    using M=dco::ga1s<dco::gt1s<float>::type>; // second-order adjoint mode
8    M::type x,y; // second-order adjoint type variables
9    dco::value(dco::value(x))=2; // set primal value of active input
10   dco::derivative(dco::value(x))=1; // seed first-order tangent
11   dco::smart_tape_ptr_t<M> tape; // enable recording of tape
12   tape->register_variable(x); // register active input
13   y=sin(x); // evaluate primal and record tape
14   tape->register_output_variable(y); // register active output
15   cout << dco::value(dco::value(y)) << " "; // get primal value of active output
16   cout << dco::derivative(dco::value(y)) << " "; // harvest first-order tangent
17   dco::derivative(y)=1; // seed first-order adjoint of active output
18   tape->interpret_adjoint(); // propagate adjoint
19   cout << dco::value(dco::derivative(x)) << " "; // harvest first-order adjoint
20   cout << dco::derivative(dco::derivative(x)) << endl; // harvest second-order adjoint
21   return 0;
22 }
```

Summary

dco/c++

Tangents

Adjoints

Second-Order Tangents

Second-Order Adjoints