Derivative Code by Overloading in C++

Introduction and Summary of Features

dco/c++ version 3.2.pre

Klaus Leppkes, and Johannes Lotz, and Uwe Naumann
The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/
Derivative Code by Overloading in C++

Introduction and Summary of Features

dco/c++ version 3.2.pre

Klaus Leppkes, and Johannes Lotz, and Uwe Naumann

Lehr- und Forschungsgebiet Informatik 12:
Software and Tools for Computational Engineering
RWTH Aachen University, Aachen, Germany
Email: info@stce.rwth-aachen.de

Abstract. dco/c++ is a flexible and efficient implementation of first- and higher-order tangent and adjoint Algorithmic Differentiation (AD) by operator overloading in C++. It combines a cache-optimized internal representation based on C++ expression templates with an intuitive and powerful application programmer interface. It also has bindings in Fortran and MATLAB. dco/c++ has been applied successfully to a number of numerical simulations in the context of, for example, large-scale parameter estimation, shape optimization, or computational finance. dco/c++ is under active development. This report provides a basic reference in form of a brief introduction to derivative code resulting from the application of AD to numerical simulation programs, a description of the four fundamental differentiation modes (first- and second-order tangent and adjoint modes, respectively) and a list of all features of the current version of dco/c++. An in-depth discussion of the latter is beyond this document. Further information on the tool can be obtained by contacting

info@stce.rwth-aachen.de.
Chapter 2

Introduction

We consider implementations of multivariate vector functions

\[ f : D^n \times D'^n \to D^m \times D'^m : (y, y') = f(x, x') \]

as computer programs over some base data type \( D \) (for example, single or higher precision floating-point data, intervals, convex/concave relaxations, vectors/ensembles).\(^1\) The \( n \) active (also: independent) and \( n' \) passive inputs are mapped onto \( m \) active (also: dependent) and \( m' \) passive outputs. The given implementation is assumed to be \( k \) times continuously differentiable at all points of interest implying the existence and finiteness of the Jacobian

\[ \nabla f = \nabla_x f(x, x') \equiv \frac{\partial y}{\partial x}(x, x') \in D^{m \times n}, \]

the Hessian

\[ \nabla^2 f = \nabla^2 f(x, x') \equiv \frac{\partial^2 y}{\partial x^2}(x, x') \in D^{m \times n \times n}, \]

if \( k \geq 2 \), and of potentially higher derivative tensors

\[ \nabla^k f = \nabla^k f(x, x') \equiv \frac{\partial^k y}{\partial x^k}(x, x') \in D^{m \times n \times \cdots \times n}, \]

We denote

\[ \nabla^k f = \left( \left[ \nabla^k f \right]_{i_1,\ldots,i_k}^{j_1,\ldots,j_k} \right)_{i=0,\ldots,m-1}^{j_1,\ldots,j_k=0,\ldots,n-1} \]

and we use \( * \) to denote the entire range of an index. For example, \( [\nabla f]_i^* \) denotes the \( i \)th row and \( [\nabla f]^*_j \) the \( j \)th column of the Jacobian, respectively. Algorithmic differentiation is implemented by overloading of elemental functions including the built-in functions and operators of C++ as well as user-defined higher-level elemental functions.

1 First-Order Tangent Mode

Generic first-order scalar tangent mode enables the computation of products of the Jacobian with vectors \( x^{(1)} \in D^n \)

\[ y^{(1)} = \nabla f \cdot x^{(1)} \in D^m \]

through provision of a generic first-order scalar tangent data type over arbitrary base data types \( D \).

\(^1\) We assume that the arithmetic inside \( f \) is completely defined (through overloading of the elemental functions; see below) for variables from \( D \).
2 First-Order Adjoint Mode

Generic first-order scalar adjoint mode enables the computation of products of the transposed Jacobian with vectors \( y^{(1)} \in D^m \)

\[
x^{(1)} = \nabla f^T \cdot y^{(1)} \in D^n
\]

through provision of a generic first-order scalar adjoint data type over arbitrary base data types \( D \).

3 Second- and Higher-Order Tangent Mode

3.1 Generic second-order scalar tangents

Instantiation of the generic first-order scalar tangent data type with the generic first-order scalar tangent data type over a non-derivative base data type yields the second-order scalar tangent data type. It enables the computation of scalar projections of the Hessian \( \nabla^2 f \in D^{m \times n \times n} \) in its two domain dimensions of length \( n \) as

\[
y^{(1,2)} = < \nabla^2 f, x^{(1)}, x^{(2)} > = \left( x^{(1)^T} \cdot \left[ \nabla^2 f \right]_{i \cdot i} \cdot x^{(2)} \right)_{i=0,\ldots,n-1} \in D^n
\]

for \( x^{(1)}, x^{(2)} \in D^n \) over arbitrary base data types \( D \). The computational cost of accumulating the whole Hessian over \( D \) becomes \( O(n^2) \cdot \text{Cost}(f) \). with both \( x^{(1)} \) and \( x^{(2)} \) ranging independently over the Cartesian basis vectors in \( D^n \).

3.2 Generic third- and higher-order tangents

Instantiation of tangent types with \( k \)th-order tangent types yields \((k+1)\)th-order tangent types.

4 Second- and Higher-Order Adjoint Mode

4.1 Generic second-order scalar adjoints

Instantiation of the generic first-order scalar adjoint data type with the generic first-order scalar tangent data type over a non-derivative base data type yields a second-order scalar adjoint data type. It enables the computation of scalar projections of the Hessian \( \nabla^2 f \in D^{m \times n \times n} \) in its image and one of its domain dimensions as

\[
x^{(2)}_{(1)} = < y^{(1)}, \nabla^2 f, x^{(2)} > = \left( y^{(1)^T} \cdot \left[ \nabla^2 f \right]_{j \cdot j} \cdot x^{(2)} \right)_{j=0,\ldots,n-1} \in D^n
\]

for \( y^{(1)} \in D^m \) and \( x^{(2)} \in D^n \) over arbitrary base data types \( D \). The computational cost of accumulating the whole Hessian over \( D \) becomes \( O(m \cdot n) \cdot \text{Cost}(f) \). with \( y^{(1)} \) and \( x^{(2)} \) ranging over the Cartesian basis vectors in \( D^m \) and \( D^n \), respectively.
4.2 Other generic second-order adjoints

Instantiation of the generic first-order vector adjoint data type with the generic first-order scalar tangent data type over a non-derivative base data type yields a second-order adjoint data type. Similarly, instantiation of the generic first-order scalar adjoint data type with the generic first-order vector tangent data type over a non-derivative base data type yields a second-order adjoint data type.

Symmetry of the Hessian in its two domain dimensions yields the following additional second-order adjoint data types: Instantiation of a generic first-order tangent data type with a generic first-order adjoint data type over a non-derivative base data type yields a second-order adjoint data type for computing

\[ < y^{(1)}_{(2)}, \nabla^2 f, x^{(1)} > = \left( y^{(1)}_{(2)} \cdot \left[ \nabla^2 f \right]_{j}^{j,*} \cdot x^{(1)} \right)_{j=0,...,n-1} \in D^n. \]

Similarly, instantiation of a generic first-order adjoint data type with a generic first-order adjoint data type over a non-derivative base data type yields a second-order adjoint data type for computing

\[ < x_{(1,2)}, y^{(1)} > = \left( y^{T}_{(1)} \cdot \left[ \nabla^2 f \right]_{j}^{j,*} \cdot x_{(1,2)} \right)_{j=0,...,n-1} \in D^n. \]

4.3 Generic third- and higher-order adjoints

Instantiation of tangent types with \( k \)-th-order adjoint types yields \((k+1)\)-th-order adjoint types. Similarly, instantiation of adjoint types with \( k \)-th-order tangent or adjoint types yields \((k+1)\)-th-order adjoint types.
Chapter 3

Features (Summary)

We list all features of the current release of dco/c++ without further explanation. Contact info@stce.rwth-aachen.de to find out more.

In blue new features in this release.

- stable
  1. generic first- and higher-order tangents (scalar mode / vector mode)
  2. generic first- and higher-order adjoints (scalar mode / vector mode)
  3. blob tape / chunk tape, file tape
  4. activity analysis for recording
  5. thread-safety by support for multiple tapes
  6. external adjoint interface, i.e. support for user-callbacks
  7. adjoint code module interface
  8. direct tape manipulation (user-defined gradients)
  9. tape compression (tape-based preaccumulation of gradients/Jacobians)
  10. Adjoint MPI support
  11. vector of adjoints separated from tape; benefits:
     • arbitrary data type for adjoint vector
     • vector and scalar modes with same data type (no re-instantiation of code base)
     • parallel use of same tape w/ multiple adjoint vectors (e.g. using OpenMP)

- experimental
  1. vector adjoint mode with multiple tape support: ga1vm
  2. adjoint code generation during overloading (full unrolling)
  3. debugging using a combined data type: finite differences vs. tangents vs. adjoints; benefits:
     • automatic check of externally implemented adjoints (tangent/adjoint identity)
     • weak discontinuity detection (using finite differences)
     • control flow discontinuity detection
  4. dco/map: meta adjoint programming (different piece of software)

- prototyped (outlook for v4.0)
  1. C++11 support
  2. code instrumentation for bidirectional dataflow analysis, debugging during tape interpretation, determination of numerical intensive kernels
  3. Linux: HPC version using low-level memory access; benefits:
     • active datatype occupies same memory as passive datatype: simple decay possible;
       zero-copy when calling passive kernels
• fixed size adjoint memory (mem(adjoints) = mem(primals)); only tape grows (sequential write/read only \(\rightarrow\) well suited for writing/reading to/from file)

4. Linux: OpenMP support w/ automatic spawning and reducing of per-thread tapes

5. Linux: fully automatic checkpointing support
Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

http://aib.informatik.rwth-aachen.de/

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: biblio@informatik.rwth-aachen.de

2013-02 Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssyste-
teme in Klein- und mittelständischen Unternehmen
2013-03 Markus Towara and Uwe Naumann: A Discrete Adjoint Model for Open-
FOAM
2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and
Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code
with Underlying Libraries
2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Es-
sentials 2013
2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination
proving through cooperation
2013-07 André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die ex-
trakorporale Lungenunterstützung
2013-08 Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika ´Abrahám: On
Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving
over the Real Numbers
2013-10 Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and
Hao Wu: Performance Analysis of Computing Servers using Stochastic
Petri Nets and Markov Automata
2013-12 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and
Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of In-
teger Programs
2013-13 Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt,
René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard
Rumpe, Dirk Thüßen, and Klaus Wehrle: SensorCloud: Towards the Inter-
disciplinary Development of a Trustworthy Platform for Globally Inter-
connected Sensors and Actuators
2013-14 Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Pro-
cedures
2013-16 Carsten Otto: Java Program Analysis by Symbolic Execution
2013-19 Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tol-
erance in the Real-Time Transport Protocol
2013-20 Jacob Palczynski: Time-Continuous Behaviour Comparison Based on
Abstract Models
2014-02 Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software
2014-03 Uwe Naumann, Klaus Leppkes, and Johannes Lotz: dco/c++ User Guide
2014-04 Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata
2014-05 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic
2014-06 Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video
2014-08 Christina Jansen, Florian Göbe, and Thomas Noll: Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs
2015-03 Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity
2015-05 Dominique Gückel: Synthesis of State Space Generators for Model Checking Microcontroller Code
2015-06 Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers’ Conference “Frontiers of Formal Methods”
2015-07 Hilal Diab: Experimental Validation and Mathematical Analysis of Cooperative Vehicles in a Platoon
2015-09 Xin Chen: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models
2015-10 Stefan Wüller, Marián Kühnel, and Ulrike Meyer: Information Hiding in the Public RSA Modulus
2015-11 Christoph Matheja, Christina Jansen, and Thomas Noll: Tree-like Grammars and Separation Logic
2015-12 Andreas Polzer: Ansatz zur variantenreichen und modellbasierten Entwicklung von eingebetteten Systemen unter Berücksichtigung regelungs- und softwaretechnischer Anforderungen
2015-13
2015-14  Niloofar Safiran and Uwe Naumann: Symbolic vs. Algorithmic Differentiation of GSL Integration Routines
2016-02  Ibtissem Ben Makhlouf: Comparative Evaluation and Improvement of Computational Approaches to Reachability Analysis of Linear Hybrid Systems
2016-03  Florian Frohn, Matthias Naaf, Jera Hensel, Marc Brockschmidt, and Jürgen Giesl: Lower Runtime Bounds for Integer Programs
2016-04  Jera Hensel, Jürgen Giesl, Florian Frohn, and Thomas Ströder: Proving Termination of Programs with Bitvector Arithmetic by Symbolic Execution
2016-06  Martin Henze, René Hummen, Roman Matzutt, Klaus Wehrle: The SensorCloud Protocol: Securely Outsourcing Sensor Data to the Cloud

* These reports are only available as a printed version.
Please contact biblio@informatik.rwth-aachen.de to obtain copies.