Adjoint Code Design Patterns

Uwe Naumann

RWTH Aachen University, Germany

QuanTech Conference, London, April 2016
Outline

Why Adjoints?

What Are Adjoints?

Software Tool Support: dco/c++

Adjoint Code Design Patterns
Why Adjoints?
Setting the Stage

In


we price a simple European Call option written on an underlying $S = S(t) : \mathbb{R}^+ \to \mathbb{R}^+$, described by the SDE

$$dS(t) = S(t) \cdot r \cdot dt + S(t) \cdot \sigma(t, S(t)) \cdot dW(t)$$

with time $t \geq 0$, maturity $T > 0$, strike $K > 0$, constant interest rate $r > 0$, volatility $\sigma = \sigma(t, S) : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$, and Brownian motion $W = W(t) : \mathbb{R}^+ \to \mathbb{R}$.

We use this simple case study for illustration of common patterns found in financial simulations in the context of *Adjoint Algorithmic Differentiation (AAD)*.
Why Adjoints?
Setting the Stage

The value of a European call option driven by $S$ is given by the expectation

$$V = \mathbb{E}e^{-r \cdot T} \cdot (S(T) - K)^+$$

for given interest $r$, strike $K$, and maturity $T$. It can be evaluated using Monte Carlo simulation or restated as a PDE and solved by a finite difference method.

We compare the computation of the gradient of the option price $V \in \mathbb{R}^+$ wrt. the problem parameters ($K, S(0), r, T$, and the variable number of parameters of the local volatility surface) by finite differences with an adjoint code based on dco/c++ ...
Why Adjoints?

Setting the Stage

- Monte Carlo method using $10^4$ paths performing 360 time steps each; gradient of size $n = 222$; running on standard PC with 3GB of RAM available

<table>
<thead>
<tr>
<th></th>
<th>primal</th>
<th>central FD</th>
<th>naive AAD</th>
<th>robust AAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time (sec.)</td>
<td>2.3</td>
<td>1010.7</td>
<td>$\infty$</td>
<td>24.4</td>
</tr>
</tbody>
</table>

- observing
  - gradient with machine accuracy by AAD
  - prohibitive memory requirement $> 40$GB for naive AAD
  - (quasi-)constant factor (here approx. 10) wrt. primal run time for AAD within given memory bounds

- aiming for robust and efficient gradient (and Hessian) code

- download adjoint option pricers from [www.nag.co.uk/downloads/addownloads](http://www.nag.co.uk/downloads/addownloads) including first- and second-order adjoint ensembles and evolutions
What Are Adjoints?

Let $y = F(x)$, $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$, be continuously differentiable. Then ...

... (approximate) tangents (directional derivatives)

$$\mathbb{R}^{m \times l} \ni \dot{Y} = \nabla F(x) \cdot \dot{X} \in \mathbb{R}^{m \times n} \in \mathbb{R}^{n \times l}$$

yield (approximate) Jacobian at $O(n) \cdot \text{Cost}(F)$;

... adjoints

$$\mathbb{R}^{n \times l} \ni \bar{X} = (\nabla F(x))^T \cdot \bar{Y} \in \mathbb{R}^{n \times m} \in \mathbb{R}^{m \times l}$$

yield Jacobian at $O(m) \cdot \text{Cost}(F')$ (cheap gradients).

Challenge: Reversal of data flow yields potentially prohibitive memory requirement ($O(\text{Cost}(F))$)!
Tangents by Overloading

\[ t := e^{\sin(x_0 \cdot x_1)} \]
\[ y_0 := x_0 \cdot t \]
\[ y_1 := \frac{t}{x_1} \]

\[ v_2 = x_0 \cdot x_1 \]
\[ \dot{v}_2 = \dot{x}_0 \cdot x_1 + x_0 \cdot \dot{x}_1 \]
\[ v_3 = \sin(v_2) \]
\[ \dot{v}_3 = \cos(v_2) \cdot \dot{v}_2 \]
\[ v_4 = \exp(v_3) \]
\[ \dot{v}_4 = v_4 \cdot \dot{v}_3 \]
\[ y_0 = x_0 \cdot v_4 \]
\[ \dot{y}_0 = \dot{x}_0 \cdot v_4 + x_0 \cdot \dot{v}_4 \]
\[ y_1 = v_4/x_1 \]
\[ \dot{y}_1 = \dot{v}_4/x_1 - v_4 \cdot \dot{x}_1/x_1^2 \]
Adjoint Code Design Patterns

5: $y_0(\ast)$

6: $y_1(\div)$

4: \(\exp\)

3: \(\sin\)

2: \(\ast\)

0: \(x_0\)

1: \(x_1\)

\(\bar{v}_2 = \bar{v}_3 = \bar{v}_4 := 0\)

\(\bar{x}_1 = \bar{v}_4/x_1^2 \cdot \bar{y}_1\)

\(\bar{v}_4+ = 1/x_1 \cdot \bar{y}_1\)

\(\bar{v}_4+ = x_0 \cdot \bar{y}_0\)

\(\bar{x}_0+ = \bar{v}_4 \cdot \bar{y}_0\)

\(\bar{v}_3+ = \bar{v}_4 \cdot \bar{v}_4\)

\(\bar{v}_2+ = \cos(v_2) \cdot \bar{v}_3\)

\(\bar{x}_1+ = x_0 \cdot \bar{v}_2\)

\(\bar{x}_0+ = x_1 \cdot \bar{v}_2\)

\(t := e^{\sin(x_0 \cdot x_1)}\)

\(y_0 := x_0 \cdot t\)

\(y_1 := \frac{t}{x_1}\)

\(\downarrow\)

// primal ...

\(\bar{v}_2 = \bar{v}_3 = \bar{v}_4 := 0\)

\(\bar{x}_1 = \bar{v}_4/x_1^2 \cdot \bar{y}_1\)

\(\bar{v}_4+ = 1/x_1 \cdot \bar{y}_1\)

\(\bar{v}_4+ = x_0 \cdot \bar{y}_0\)

\(\bar{x}_0+ = \bar{v}_4 \cdot \bar{y}_0\)

\(\bar{v}_3+ = \bar{v}_4 \cdot \bar{v}_4\)

\(\bar{v}_2+ = \cos(v_2) \cdot \bar{v}_3\)

\(\bar{x}_1+ = x_0 \cdot \bar{v}_2\)

\(\bar{x}_0+ = x_1 \cdot \bar{v}_2\)
Software Tool Support: dco/c++

**dco/c++ (derivative code by overloading in C++)** features

- tangents and adjoints of arbitrary order through recursive template instantiation for numerical simulation code implemented in C++
- front-ends for Fortran and C# (beta)
- optimized assignment-level gradient code through expression templates
- cache-optimized internal representation in various incarnations
- vector modes / detection and exploitation of sparsity
- external adjoint / Jacobian interfaces
- user-defined intrinsics
- intrinsic NAG Library functions (e.g. Linear Algebra, Interpolation, Root Finding, Nearest Correlation Matrix)
- support for parallelism: thread-safe data structures, adjoint MPI library, GPU coupling
Moreover, we have implemented an industrial-strength software engineering infrastructure\(^1\) including:

- professional software development and quality assurance (NAG ↔ STCE)
- multi-platform (Windows, Linux, Mac OS X) / compiler (vcc, gcc, clang, icc) support
- overnight building / testing
- version control
- user documentation / examples
- numerous successful applications, including QuantLib, OpenFOAM, PETSc, ICON, Telemac, Jurassic2, ACE+, and several confidential projects with tier-1 investment banks

\(^1\)building on academic prototype
An algorithmic adjoint of $F : \mathbb{R}^n \to \mathbb{R}^m : y \equiv z^q = F(x)$ computes

$$\bar{X} \in \mathbb{R}^{n \times l} = \nabla F^T, \quad \bar{Y} \in \mathbb{R}^{m \times l} = \nabla F^T \bar{Z}^{k-1} \left( \nabla F^T (z^{k-1}) \left( \nabla F^T (z^{k-1}) \right) \cdots \nabla F^T (z^q) \nabla F^T (\bar{Y}) \right) \cdots$$

for $x \equiv z^0$ and based on adjoint code modules (AC Modules) $\bar{Z}^{i-1} = \nabla F^T \bar{Z}^i$ for $i = q, \ldots, 1$.

The art of differentiating computer programs / algorithmic differentiation (AD) is about provision of a correct, robust, efficient, scalable, flexible ACM module hierarchy.
Adjoint Code Design with dco/c++

Mind the Gap

Context:
\[ v = G(u) \]
\[ u_G = \nabla G(u)^T v_G \]

\[ y = F(x) \]
\[ x_G = \nabla F(x)^T y_G \]

\[ kx \]
\[ ky \]
\[ \text{evaluate - augmented - primal()} \]

\[ lx_{(x)} \]
\[ ly_{(y)} \]
\[ \text{evaluate - adjoint()} \]

\[ u_G \]
\[ x_G \]
\[ v_G \]
\[ y_G \]
We consider a Euler-Maruyama scheme performing $\mu$ explicit Euler integration steps along $\nu$ independent Monte Carlo sample paths.

A basic implementation of this method evaluates a function of the following structure:

$$y = F(x) = G\left(\begin{pmatrix} P^0_{\mu-1}(P^0_{\mu-2}(\ldots P^0_0(S^0_0(x)) \ldots)) \\ \vdots \\ P^{\nu-1}_{\mu-1}(P^{\nu-1}_{\mu-2}(\ldots P^{\nu-1}_0(S^{\nu-1}_0(x)) \ldots)) \end{pmatrix}\right),$$

where the arguments $x$ (strike price, time to maturity, interest rate(s), price of underlying and market observed volatilities) are scattered over the individual paths by the $S^i$, $i = 0, \ldots, \nu - 1$, each path performs integration steps $P^i_j$, $j = 0, \ldots, \mu - 1$, and the results are gathered by $G$ to yield $y$ (option price).
Adjoint Code Design Patterns with dco/c++

Case Study: ACModule Class Diagram
Adjoint Code Design Patterns with dco/c++
Case Study: ACModule Object Diagram

Further Scenarios

- Symbolic Adjoints of Implicit Functions
  - Adjoint Linear Systems
  - Adjoint Nonlinear Systems
  - Adjoint Nonlinear (Locally) Convex Objective Functions

- Local Finite Differences
  - Black-Box Primal
  - Active Control Flow Dependence
  - Nonsmooth Time Stepping

- Preaccumulation of Local Jacobians
  - Detection and Exploitation of Sparsity
  - Elimination Techniques on Linearized DAGs

- Reverse Accumulation for Adjoint Fix-Point Iteration

- Dynamic Call Tree Reversal

- Verified Enclosures through Interval Arithmetic / Convex Envelopes

- ...

Naumann, April 2016, Adjoint Code Design Patterns
Adjoint Code Design Patterns with dco/c++

Selected Scenarios (Toy)

\[ x^2 - p = 0 \]

differentiate

\[ \bar{x} \cdot \frac{d}{dp} \]

\[ \bar{x} \cdot \left( 2 \cdot x \cdot \frac{dx}{dp} - 1 \right) = 2 \cdot x \cdot \left( \bar{x} \cdot \frac{dx}{dp} \right) - \bar{x} = 0 \]

perturb

\[ \dot{x}^* = S(x^0, p + h) \]

\[ \tilde{p}^* = \tilde{S}(\tilde{p}^0, x^*, \bar{x}) \]

interpret

\[ x^* \approx \sqrt{p} \]

\[ \bar{p} = \frac{\bar{x}}{2 \sqrt{p}} = \frac{\bar{x}}{2 \cdot x} \approx \tilde{p}^* \approx \tilde{p}^* \approx \bar{x} \cdot \frac{x^* - x^*}{h} \]

\[ (x^*, G) = \rightarrow S(x^0, p) \]

record

\[ \bar{x} \cdot \frac{d}{dp} \]

\[ \bar{x} \cdot \frac{d}{dp} \]

\[ \bar{x} \cdot \frac{d}{dp} \]

\[ \bar{x} \cdot \frac{d}{dp} \]

Adjoint Code Design Patterns with dco/c++

Local Finite Differences (Notation: $\bar{v} \equiv v^{(1)}$)

Context:

\[ v = G(u) \]
\[ u_{(1)} \cdot = D G(u)^{T} \cdot v_{(1)} \]

AC Local Finite Differences

\[ y := F(x) \]

\[ y := F(x) \rightarrow x \cdot DF(x) \]

\[ x_{(1)} + \left( x \cdot DF(x) \right)^{T} \cdot y_{(1)} \]
Adjoint Code Design Patterns with dco/c++

\[ A(p) \cdot x(p) = b(p) \Rightarrow A^T \cdot \bar{b} = \bar{x} ; \quad \bar{A} = -\bar{b} \cdot x^T \]
Adjoint Code Design Patterns with dco/c++

\[ F(x(p), p) = 0 \Rightarrow \nabla_x F^T \cdot z = -\bar{x}; \quad \bar{p} = \nabla_p F^T \cdot z \]
argmin_x \ f(x(p), p) \Rightarrow \nabla_{xx} f^T \cdot z = -\bar{x}; \quad \bar{p} = \nabla_{xp} f^T \cdot z
Conclusion

The quality of an adjoint AD solution / tool is defined by

- **robustness** wrt. language features of target code
- **efficiency** of adjoint propagation
- **flexibility** wrt. design scenarios
- **sustainability** wrt. dynamics in user requirements, personnel, hard- and software

You want (sooner or later)

- top-quality AAD tools and documentation, e.g., dco/c++
- adjoint numerical libraries, e.g., NAG AD Library
- training, e.g., 1-3 day in-house courses
- support / consultancy, e.g., responsive and competent help desk
Outlook

- dco/c++ v4.x
  - tape compression / decomposition / debugging
  - run time code generation
  - file / MPI tapes
  - “heapless” taping on GPUs
- fully automatic checkpointing / adjoints
- exploratory spawning for globalization and nonsmoothness
- multi-core taping
- approximate / “gut” computing
Follow-up

naumann@stce.rwth-aachen.de