Algorithmic Differentiation of Nonsmooth and Discontinuous Functions

Jonathan Hüser and Uwe Naumann
Software and Tools for Computational Engineering
RWTH Aachen University

Abstract

Adjoint algorithmic differentiation (AAD) is exact up to machine precision and does not capture sensitivity to nearby nonsmoothness or discontinuities. Smoothing the indicator function produces a regularization effect similar to bumping the input while maintaining the efficiency of AAD. The operator-overloading tool dco/c++ supports smoothing through extensible adjoint code patterns for nonsmooth and discontinuous functions.

Adjoint Algorithmic Differentiation

- For $y = f(\mathbf{x})$ with $f : \mathbb{R}^n \to \mathbb{R}$ efficiently compute $\partial f(\mathbf{x})/\partial \mathbf{x}$

AAD	bumping
$O(1) \cdot \mathrm{cost}(f)$	$O(n) \cdot \mathrm{cost}(f)$

- dco/c++ is an operator-overloading AAD tool
  → partial derivatives are stored in tape during computation
  → no separate maintenance of primal and derivative code required

Nonsmooth and Discontinuous Functions

Examples

- Vanilla call payoff (nonsmooth)

$$P(S,K) = \begin{cases} S-K & \text{if } S-K>0 \\ 0 & \text{otherwise .} \end{cases}$$

- Digital payoff (discontinuous)

$$P(S,K) = \begin{cases} 100 & \text{if } S-K>0 \\ 0 & \text{otherwise .} \end{cases}$$

- Nonsmooth and discontinuous functions usually piecewise defined

$$f(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) & \text{if } g(\mathbf{x})>0 \\ f_2(\mathbf{x}) & \text{otherwise .} \end{cases}$$

- Derivatives near $g(\mathbf{x}) = 0$ can be challenging (e.g. Monte Carlo)

$$\frac{\partial}{\partial \mathbf{x}} \mathbb{E}\left[f(\mathbf{x},\mathbf{z})\right] \not\approx \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x},\mathbf{z}^i),\ z_1 \ldots z_{n_z} \sim \mathcal{N}(0,1) .$$

- (Stochastic) indicator normal form (INF)

$$f(\mathbf{x}) = \sum_{i=1}^{n_f} \prod_{j=1}^{n_{g,i}} \mathbb{1}\left[g_{i,j}(\mathbf{x},\mathbf{z})>0\right] \cdot f_i(\mathbf{x},\mathbf{z}) .$$

→ Tool-based generation from control flow branching programs

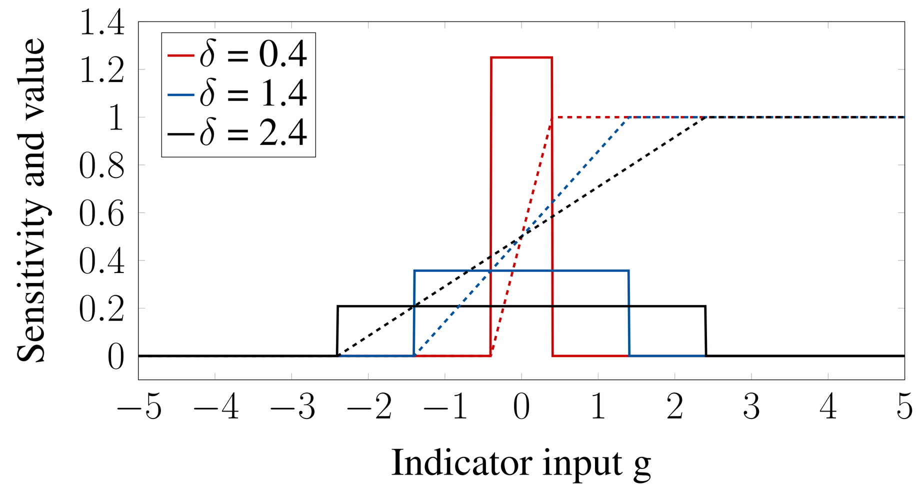
Function Regularization

- Replace indicator by continuous / smooth approximation
- Bandwidth parameter δ controls approximation error (bias)

Uniform Distribution (Call Spread)

$$\mathbb{1}[g>0] \approx \int_{-\infty}^{\infty} \mathbb{1}[g>0] \cdot \mathbb{1}[-\delta<g<\delta]\,\mathrm{d}g$$

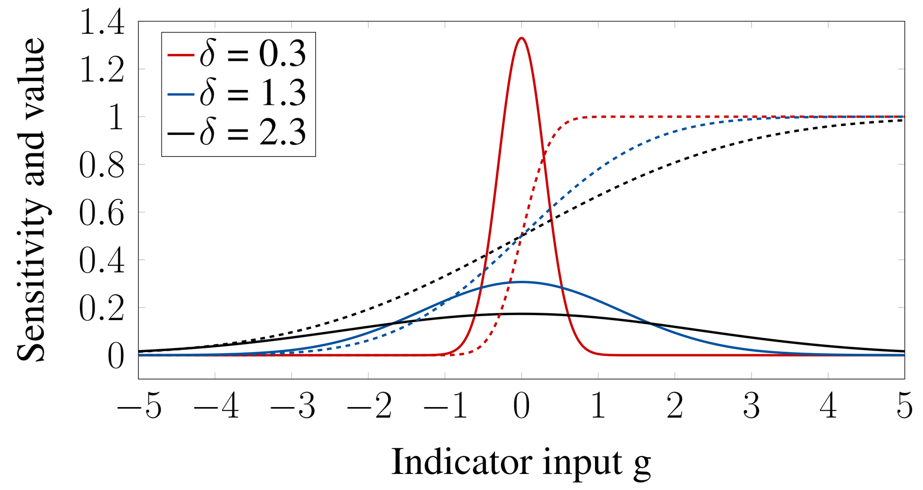
$$\frac{\partial}{\partial g}\mathbb{1}[g>0] \approx \mathbb{1}[-\delta<g<\delta] \cdot \frac{1}{2\delta} .$$



Normal Distribution

$$\mathbb{1}[g>0] \approx \int_{-\infty}^{\infty} \mathbb{1}[g>0] \cdot \frac{1}{\delta\sqrt{2\pi}} \exp\left(-\frac{g^2}{2\delta^2}\right) \mathrm{d}g$$

$$\frac{\partial}{\partial g}\mathbb{1}[g>0] \approx \frac{1}{\delta\sqrt{2\pi}} \exp\left(-\frac{g^2}{2\delta^2}\right) .$$



- Identical tape used to evaluate multiple regularizations
- Tape can evaluate $\partial g_{i,j}(\mathbf{x},\mathbf{z})/\partial \mathbf{z}$ to calibrate bandwidth δ
- Other methods for nonsmooth and discontinuous functions
  → Differentiable univariate quadrature (for stochastic case)
  → Direct evaluation of piecewise linearization

Barrier Option Monte Carlo (Case Study)

- Payoff for discretized path given in stochastic INF

$$P(S_0,K,B,r,\sigma,\mathbf{Z}) = \prod_{i=1}^{N} \mathbb{1}[B-S_i>0] \cdot \mathbb{1}[S_N-K>0] \cdot (S_N-K)$$

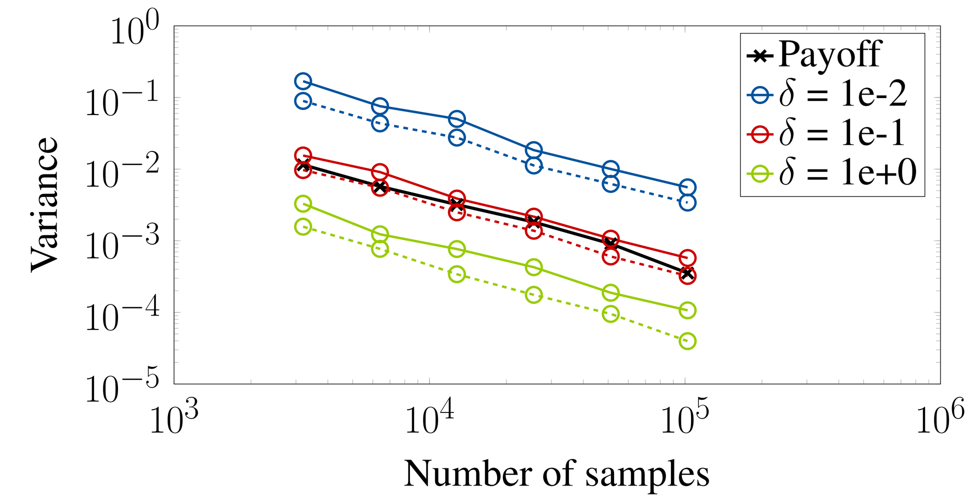
with $S_i = S_i(S_0, r, \sigma, \mathbf{Z})$.

Euler-Maruyama Path with dco/c++

```
dco::ga1s<double>::global_tape->register_variable(S0);
dco::ga1s<double>::global_tape->register_variable(K);
dco::ga1s<double>::global_tape->register_variable(B);
dco::ga1s<double>::global_tape->register_variable(r);
dco::ga1s<double>::global_tape->register_variable(sigma);

S[0] = S0;
for (int i = 1; i < N; ++i) {
  S[i] = S[i-1] + S[i-1]*r*dt + S[i-1]*sigma*sqrt(dt)*Z[i-1];
}

P = 1.0;
for (int i = 0; i < N; ++i) {
  P *= indicator(B - S[i]);
}
P *= indicator(S[N-1] - K) * (S[N-1] - K);

dco::derivative(P) = 1.0;
dco::ga1s<double>::global_tape->interpret_adjoint();

cout << "dPdS0 (Delta) " << dco::derivative(S0) << endl;
cout << "dPdK " << dco::derivative(K) << endl;
cout << "dPdB " << dco::derivative(B) << endl;
cout << "dPdr " << dco::derivative(r) << endl;
cout << "dPdsigma " << dco::derivative(sigma) << endl;
```

Regularized Monte Carlo Delta Estimator

- $S_0 = 100$, $K = 80$, $B = 110$, $r = 0$, $\sigma = 0.2$, $T = 1$
- Variance with different number of samples, $N = 1000$
- Uniform distribution (solid), normal distribution (dashed)



- Delta with $2.048e7$ samples, $N = 1000$

δ	1e-2	1e-1	1e0
uniform	-0.2345	-0.2398	-0.2397
normal	-0.2365	-0.2395	-0.2389

- Low probability events → Variance reduction still necessary

Nearest Correlation Matrix (Case Study)

- Pairwise correlations can be inconsistent
  → Find nearest $n \times n$ correlation matrix that is positive definite
- A_+ projection $\lambda_i := \max(0, \lambda_i)$ of eigenvalues, $A = Q \cdot \mathrm{diag}\,(\lambda) \cdot Q^T$

$$(A_+)_{i,j} = \sum_{k=1}^{n} \mathbb{1}[\lambda_k>0] \cdot \lambda_k \cdot Q_{i,k} \cdot Q_{j,k} .$$

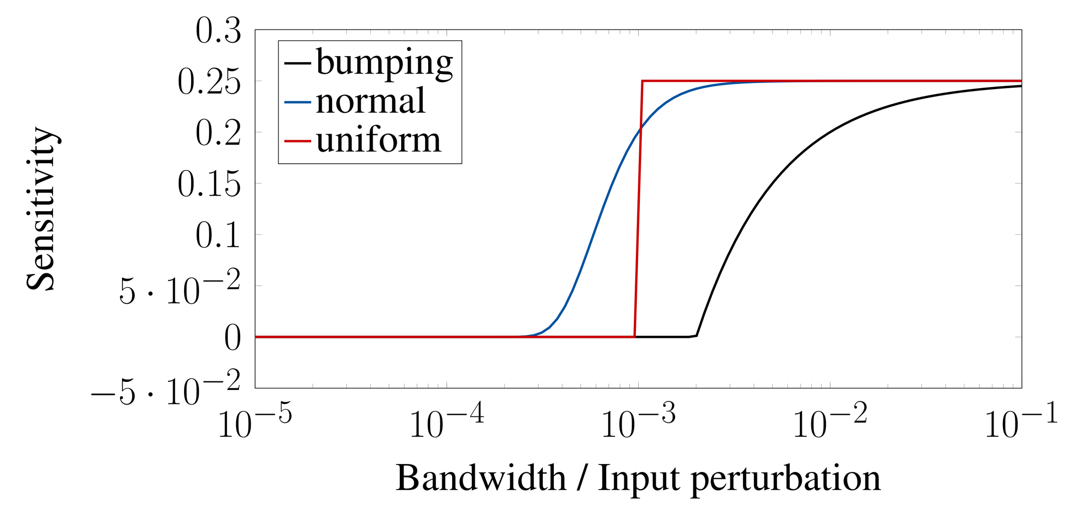
- Convex optimization problem → use implicit function theorem

Regularized Projection

- Test matrix positive definite but numerically singular

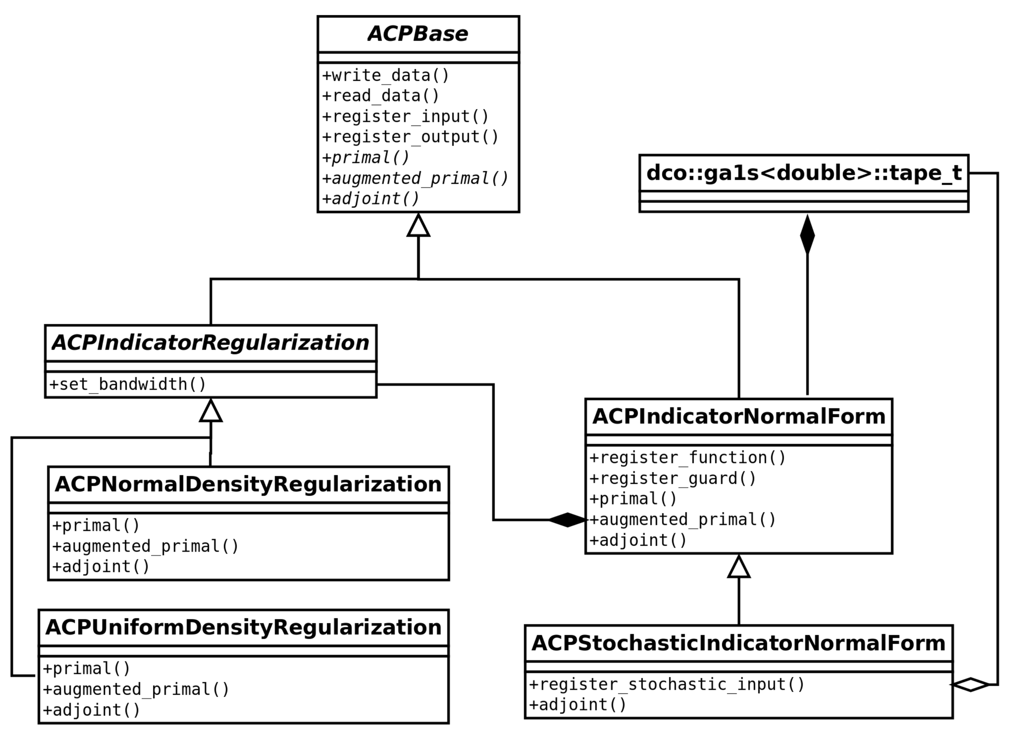
$$A = \begin{pmatrix} 1 & 0.999 & 0 \\ 0.999 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- NCM algorithms maps A to itself, no AAD sensitivities
- Regularization gives sensitivity on implicit function theorem
- Projection sensitivity $\partial(A_+)_{1,1}/\partial A_{2,1}$ for different δ



- Regularization is local bumping (shift)
  → Calibration through correlation uncertainty

Adjoint Code Patterns in dco/c++



Conclusions

- Monte Carlo sensitivities for discontinuous payoffs
- Smoothing of auxiliary functions and implicit function theorem
- Local sensitivity enables bandwidth calibration
- dco/c++ supports extensible adjoint code patterns for regularization

Contact Information:
LuFG Informatik 12: STCE
RWTH Aachen University
D-52056 Aachen, Germany
Phone: +49 (0)241 80 29224
Email: hueser@stce.rwth-aachen.de