Sacado: Automatic Differentiation Tools for C++ Codes

Eric Phipps
etphipp@sandia.gov
David Gay, Ross Bartlett

Trilinos User Group 2009

SAND 2009-7540C
What is Automatic Differentiation (AD)?

• Technique to compute analytic derivatives without hand-coding the derivative computation

• How does it work -- freshman calculus
  – Computations are composition of simple operations (+, *, sin(), etc...) with known derivatives
  – Derivatives computed line-by-line, combined via chain rule

• Derivatives accurate as original computation
  – No finite-difference truncation errors

• Provides analytic derivatives without the time and effort of hand-coding them

\[ y = \sin(e^x + x \log x), \quad x = 2 \]

\[ \begin{array}{|c|c|c|}
\hline
\text{ } & x & \frac{d}{dx} \\
\hline
x & 2.000 & 1.000 \\
\hline
t & e^x & 7.389 & 7.389 \\
\hline
u & \log x & 0.301 & 0.500 \\
\hline
v & xu & 0.602 & 1.301 \\
\hline
w & t + v & 7.991 & 8.690 \\
\hline
y & \sin w & 0.991 & -1.188 \\
\hline
\end{array} \]
Sacado: AD Tools for C++ Codes

• Sacado implements AD via operator overloading and C++ templating
  – Template your code on scalar type (double --> ScalarT)
  – Instantiate template code on Sacado AD types to get derivatives
  – Expression templates for OO efficiency

• Sacado provides several modes of Automatic Differentiation (AD)
  – Forward (Jacobians, Jacobian-vector products, …)
  – Reverse (Gradients, Jacobian-transpose-vector products, …)
  – Taylor (High-order univariate Taylor series)
  – Sacado is itself templated on the scalar type to allow nesting of modes (higher derivatives)
  – Embedded Stochastic Galerkin methods with Stokhos

• Designed for use in large-scale C++ codes
  – Apply AD at “element-level” for dense element derivatives
  – Very successful in Sandia application codes
  – Sacado::FEApp example demonstrates approach

• Sacado provides other useful utilities
  – Scalar flop counting (Ross Bartlett)
  – Scalar parameter library
  – Template utilities and basic MPL
#include "Sacado.hpp"

// The function to differentiate
template <typename ScalarT>
ScalarT func(const ScalarT& a, const ScalarT& b, const ScalarT& c) {
    ScalarT r = c*std::log(b+1.)/std::sin(a);
    return r;
}

int main(int argc, char **argv) {
    double a = std::atan(1.0);  // pi/4
    double b = 2.0;
    double c = 3.0;
    int num_deriv = 2;  // Number of independent variables

    // Fad objects
    Sacado::Fad::DFad<double> afad(num_deriv, 0, a);  // First (0) indep. var
    Sacado::Fad::DFad<double> bfad(num_deriv, 1, b);  // Second (1) indep. var
    Sacado::Fad::DFad<double> cfad(c);  // Passive variable
    Sacado::Fad::DFad<double> rfad;  // Result

    // Compute function
    double r = func(a, b, c);

    // Compute function and derivative with AD
    rfad = func(afad, bfad, cfad);

    // Extract value and derivatives
    double r_ad = rfad.val();  // r
    double drda_ad = rfad.dx(0);  // dr/da
    double drdb_ad = rfad.dx(1);  // dr/db
New Features for Trilinos 10

- Complex variable support
- Teuchos::ScalarTraits support
  - Allows differentiation of generic Teuchos::BLAS implementations
- Vector forward derivative objects
  - Value & derivatives stored contiguously
- Custom forward-mode differentiated BLAS
  - Improved derivative performance for BLAS operations