SAND2012-9161P

LAMENT

Applying Sacado to a Sierra Material Model Library

Funding: ASC Algorithms (Hoekstra)

David Littlewood

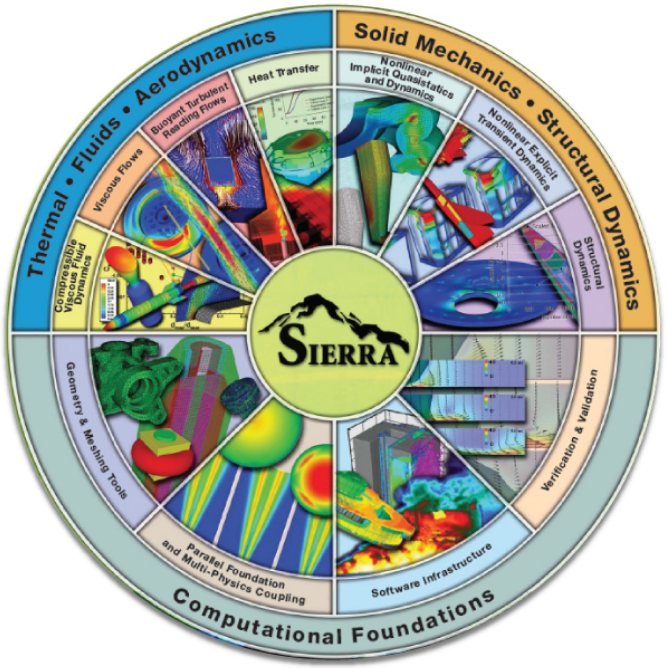
Albany Developers Meeting

2 October 2012

Sierra

Sierra is the engineering mechanics simulation code suite supporting the nation's nuclear weapons mission, as well as other customers

- Advanced Simulation and Computing (ASC) code
- Developed and maintained within center 1500
- Customers include DOE, DoD, and industrial partners
- Capabilities:
  - Solid Mechanics (Presto, Adagio)
  - Structural Dynamics (Salinas)
  - Thermal / Fluids (Aria)

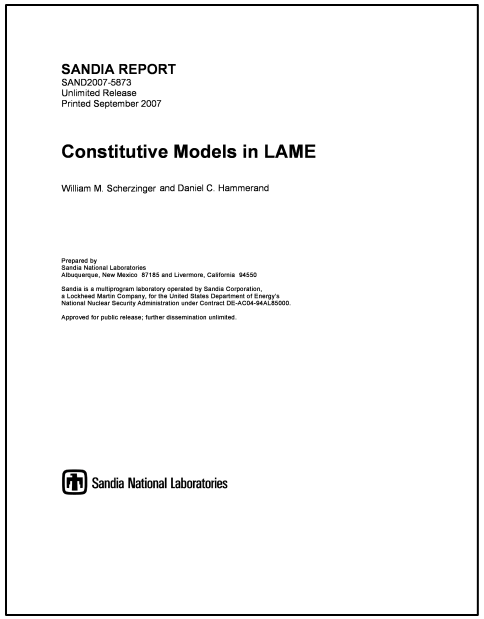


Sandia National Laboratories

Library of Advanced Materials for Engineering (LAME)

LAME IS THE MATERIAL LIBRARY UTILIZED BY SIERRA/SOLIDMECHANICS

- 80+ material models
- ~30 “supported” material models for Sierra/SolidMechanics
- Material models for solid elements, shell, cohesive zones, …
- Equation of State (EOS) material models
- Examples of material models ported to LAME:
  - EPIC material models (DoD code)
  - Abaqus User Material (UMAT) models



SANDIA REPORT
SAND2007-5873
Unlimited Release
Printed September 2007

Constitutive Models in LAME

William M. Scherzinger and Daniel C. Hammerand

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

LAME Interface

- Input
  - Kinematic variables (strain, strain rate, volume, …)
  - Material model state variables
- Output
  - Stress
  - Updated material model state variables
- Standardized C interface
  - Wraps model-specific interfaces with a common interface
  - Many material models written in Fortran
    - Extreme example: Kayenta model contains 26,738 lines of Fortran
- Sierra/SolidMechanics calls LAME at the element level
  - Kinematic calculations are specific to each element type
  - Hierarchy of derived classes, nested classes, etc.
  - For historical reasons, LAME wrapped in several layers of abstraction
- Albany calls LAME from within the `LameStress` evaluator
  - Deformation gradient is required field
  - Stress is evaluated field
  - Currently limited to small subset of LAME material models

LAME Interface

INTERFACE INCLUDES POINTERS TO A LARGE SET OF DATA

```
struct matParams {

  int nelements;
  int nnodes_per_elem;
  int nintg;
  double dt;
  double time;
  double dt_mat;
  double * strain_rate;
  double * spin;
  double * stress_old;
  double * stress_new;
  double * state_old;
  double * state_new;
  std::vector<double *> sub_states_old;
  std::vector<double *> sub_states_new;
  double * temp_old;
  double * temp_new;
  double * capillarypressure_old;
  double * capillarypressure_new;
  double * wettingphasepressure_old;
  double * wettingphasepressure_new;
  double * nonwettingphasesaturation_old;
  double * nonwettingphasesaturation_new;
  double * concentration;
  double * left_stretch;
  double * rotation;
  double * volume;
  double * entropy;
  double * energy_balance_term;
  double * material_properties;
  double * tangent_moduli;
  double * centroid;
  double * ym_old;
  double * ym_new;
  double * nU_new;
  double * nU_old;
  double * bulk_scaling;
  double * shear_scaling;
  double * sound_speed_old;
  double * sound_speed_new;
  double * density_old;
  double * density_new;
  double * bulk_viscosity_old;
  double * bulk_viscosity_new;
  double * internal_energy_old;
  double * internal_energy_new;
  double * bulk_modulus;
  double * shear_modulus;
  double * thickness;
  double * base_vectors;
  double * characteristic_length;
  double * yield_stress;
  double * equivalent_plastic_strain_old;
  double * equivalent_plastic_strain_new;
  double * plastic_strain_rate;
  double * element_global_base_vectors;
  double * element_global_coordinates;
  double * scratch;
  std::string model;
  bool probingElement;
  double * strain_rate_avg;
  double * crack_flag_old;
  double * crack_flag_new;
  double * decay_old;
  double * decay_new;
  double * failure_measure_old;
  double * failure_measure_new;

};
```

Sacado: An Automatic Differentiation Library

WHAT IS THE SACADO PACKAGE?

Sacado is a C++ template-based toolset for automatic differentiation (AD) and sensitivity analysis [Phipps, Gay]

HOW DOES IT WORK?

- Computer-science wizardry utilizing C++ templates and operator overloading
- Targeted code is templated on the scalar type (e.g., double)
  - When instantiated for the standard type, original code is recovered
  - When instantiated for an AD type, Sacado functionality is enabled
- Sacado's AD types contain:
  - A base value (the original scalar)
  - Additional fields corresponding to derivative terms

WHAT ARE THE APPLICATIONS TO SIERRA/SM?

- Automatic calculation of function derivatives
- Applicable to tangent matrix for implicit time integration and/or modal analysis
- Ability to carry out multiple function evaluations simultaneously (multiprobe)

A Simple Example of Automatic Differentiation with Sacado

FUNCTION TEMPLATED ON THE SCALAR TYPE

- Standard C++ template syntax
- Replace scalar (double) with template argument ScalarT

```
template<typename ScalarT> void testFunction(ScalarT* a,
                                             ScalarT* b)
{
  a[0] = 2.0 + 3.0*b[0] + 4.0*b[1]*b[1];
}
```

EXPLICIT TEMPLATE INSTANTIATION

- Define specific instances of templated function in *.cpp file
- Avoids need to use header files exclusively
- Improved compilation time
- Loss of generality (cannot use arbitrary template argument elsewhere in code)

```
template void testFunction<double>(double* a,
                                   double* b);

template void testFunction<Sacado::Fad::DFad<double> >(Sacado::Fad::DFad<double>* a,
                                                       Sacado::Fad::DFad<double>* b);
```

A Simple Example of Automatic Differentiation with Sacado

FUNCTION INSTANTIATED WITH TEMPLATE ARGUMENT = double

```
vector<double> a(1);
vector<double> b(2);
b[0] = 2.0;
b[1] = 3.0;

testFunction(&a[0], &b[0]);

cout << "a[0] = " << a[0] << endl;
```

```
a[0] = 44
```

FUNCTION INSTANTIATED WITH TEMPLATE ARGUMENT = Sacado::Fad::DFad<double>

```
vector<Sacado::Fad::DFad<double> > a(1);
vector<Sacado::Fad::DFad<double> > b(2);
b[0] = Sacado::Fad::DFad<double>(2, 0, 2.0);
b[1] = Sacado::Fad::DFad<double>(2, 1, 3.0);

testFunction(&a[0], &b[0]);

cout << "a[0].val() = " << a[0].val() << endl;
cout << "a[0].dx(0) = " << a[0].dx(0) << ", a[0].dx(1) = " << a[0].dx(1) << endl;
```

```
a[0].val() = 44
a[0].dx(0) = 3, a[0].dx(1) = 24
```

LAMENT: LAME Now Templated

GOAL: OBTAIN DERIVATIVES OF STRESS COMPONENTS W.R.T. NODAL DISPLACEMENTS

```
template <typename ScalarT>
struct matParams {

   int nelements;
   int nnodes_per_elem;
   int nintg;
   double dt;
   double time;
   double dt_mat;
   ScalarT * strain_rate;
   ScalarT * spin;
   double * stress_old;
   ScalarT * stress_new;
   double * state_old;
   double * state_new;
   std::vector<double *> sub_states_old;
   std::vector<double *> sub_states_new;
   double * temp_old;
   double * temp_new;
   double * capillarypressure_old;
   double * capillarypressure_new;
   double * wettingphasepressure_old;
   double * wettingphasepressure_new;
   double * nonwettingphasesaturation_old;
   double * nonwettingphasesaturation_new;
   double * concentration;
   ScalarT * deformation_gradient
   ScalarT * left_stretch;
   ScalarT * rotation;
   double * volume;
   double * entropy;
   double * energy_balance_term;
   double * material_properties;
   double * tangent_moduli;
   double * centroid;
   double * ym_old;
   double * ym_new;
   double * nU_new;
   double * nU_old;
   double * bulk_scaling;
   double * shear_scaling;
   double * sound_speed_old;
   double * sound_speed_new;
   double * density_old;
   double * density_new;
   double * bulk_viscosity_old;
   double * bulk_viscosity_new;
   double * internal_energy_old;
   double * internal_energy_new;
   double * bulk_modulus;
   double * shear_modulus;
   double * thickness;
   double * base_vectors;
   double * characteristic_length;
   double * yield_stress;
   double * equivalent_plastic_strain_old;
   double * equivalent_plastic_strain_new;
   double * plastic_strain_rate;
   double * element_global_base_vectors;
   double * element_global_coordinates;
   double * scratch;
   std::string model;
   bool probingElement;
   double * strain_rate_avg;
   double * crack_flag_old;
   double * crack_flag_new;
   double * decay_old;
   double * decay_new;
   double * failure_measure_old;
   double * failure_measure_new;

};
```

Unit Test: Calculation of Stress

```
// Instantiate the material model and the material parameters structure

Material<double> * elasticMat = new ElasticNew<double>(*props);
matParams<double> * matp = new matParams<double>();

matp->nelements      = 1;
matp->dt             = 5e-3;
double strainRate[]  = { 0.025, 0.0, 0.0, 0.0, 0.0, 0.0 };
double stressOld[]   = { 0.0,   0.0, 0.0, 0.0, 0.0, 0.0 };
double stressNew[]   = { 0.0,   0.0, 0.0, 0.0, 0.0, 0.0 };
matp->strain_rate    = strainRate;
matp->stress_old     = stressOld;
matp->stress_new     = stressNew;

// Compute the stress

elasticMat->getStress(matp);

// Verify computed stress values

double stressNewExp[] = { 5e-3*0.025*youngsValue, 0.0, 0.0, 0.0, 0.0, 0.0 };

for(int i=0; i< 6; i++)
     ASSERT_NEAR(stressNewExp[i], stressNew[i], 1e-14);
```

Unit Test: Calculation of Stress and Derivatives w.r.t. Strain

```
typedef Sacado::ELRFad::DFad<double> FadType

// Instantiate the material model and material parameters structure

Material<FadType> * elasticMat = new ElasticNew<FadType>(*props);
matParams<FadType> * matp = new matParams<FadType>();

matp->nelements    = 1;
matp->dt           = 5.0e-3;
double stressOld[] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
matp->stress_old   = stressOld;

// the components of the strain rate are the independent variables
int numDof = 6;
double strainRate[] = { 0.025, -0.01, -0.02, 0.0001, -0.0003, 0.002 };
vector<FadType> strainRate_AD(numDof);
for(int i=0 ; i<numDof ; ++i){
  strainRate_AD[i].diff(i, numDof);
  strainRate_AD[i].val() = strainRate[i];
}
matp->strain_rate = &strainRate_AD[0];

// the components of the stress tensor (stressNew) are the dependent variables
double stressNew[] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 };
vector<FadType> stressNew_AD(numDof);
for(int i=0 ; i<numDof ; ++i){
  stressNew_AD[i].val() = stressNew[i];
}
matp->stress_new = &stressNew_AD[0];

// Compute the stress

elasticMat->getStress(matp);
```

Unit Test: Calculation of Stress and Derivatives w.r.t. Strain

```
// K = bulk modulus
// mu = shear modulus

elasticityTensor[0][0] = K+4.0*mu/3.0;
elasticityTensor[0][1] = K-2.0*mu/3.0;
elasticityTensor[0][2] = K-2.0*mu/3.0;
elasticityTensor[1][0] = K-2.0*mu/3.0;
elasticityTensor[1][1] = K+4.0*mu/3.0;
elasticityTensor[1][2] = K-2.0*mu/3.0;
elasticityTensor[2][0] = K-2.0*mu/3.0;
elasticityTensor[2][1] = K-2.0*mu/3.0;
elasticityTensor[2][2] = K+4.0*mu/3.0;
elasticityTensor[3][3] = 2.0*mu; // factor of two for Mandel notation
elasticityTensor[4][4] = 2.0*mu;
elasticityTensor[5][5] = 2.0*mu;

// Verify the stress calculation

for(int i=0; i< 6; i++)
  ASSERT_NEAR(stressNew_AD[i].val(), stressNewExpected[i], tol);

// verify the Jacobian

for(int i=0; i<6; ++i) {
  for(int j=0 ; j<6; ++j)
    ASSERT_NEAR(stressNew_AD[i].dx(j), elasticityTensor[i][j]*timeStep, tol);
  }
}
```

Albany Tests

TESTS THAT CALL LAME

- LameStaticElasticity3D
  - Case 1) Matrix-free solver
  - Case 2) Tangent constructed via finite-difference probe
- LameMultiMaterials
  - Tests ability to assign a large number of materials to different blocks

TESTS THAT CALL LAMENT

- LamentStaticElasticity3D
  - Case 1) Tangent constructed via automatic differentiation

```
<ParameterList name="Problem">
  <Parameter name="Name" type="string" value="Lame"/>
  <Parameter name="Lame Material Model" type="string" value="Elastic_New"/>
  <ParameterList name="Lame Material Parameters">
    <Parameter name="Youngs Modulus" type="double" value="1.0"/>
    <Parameter name="Poissons Ratio" type="double" value="0.25"/>
  </ParameterList>

  . . .

</ParameterList>
```

Ongoing Work

LAMENT

- Port additional LAME materials to LAMENT
- Tackle a material model written in Fortran
  Additional unit tests

ALBANY

- Explore power of Sacado beyond tangent stiffness matrix
- Additional system tests

SIERRA/SOLIDMECHANICS

Proof-of-concept demonstration of tangent matrix construction via automatic differentiation

- Templatize code spanning from probing operation through LAME material models
  1. Element tangent probe
  2. Internal force calculator
     - Nested class hierarchy within large code base adds complexity
     - Utilize new master element that wraps Intrepid [Ostien]
  3. Material manager
  4. LAME interface
  5. LAME material model(s)
- Milestone #1: Compute finite-difference tangent with multiprobe type
- Milestone #2: Compute tangent via automatic differentiation

Questions?

LAMENT

Applying Sacado to a Sierra Material Model Library

David Littlewood

djlittl@sandia.gov

Multiphysics Simulation Technologies (Org. 1444)