Automatic Differentiation
Algorithms and Data Structures

Chen Fang
PhD Candidate
Joined: January 2013
FuRSST Inaugural Annual Meeting
May 8th, 2014
About \( \frac{2}{3} \) of the code for physics in simulators computes derivatives.
Automatic Differentiation (AD)

1. AD data type

\[ \tilde{x}_1 \rightarrow [x_1, \left( \frac{\partial \tilde{x}_1}{\partial x_1}, \frac{\partial \tilde{x}_1}{\partial x_2} \right) ] \]

2. Operator overloading

\[ \nabla (x \ast y) = x\nabla y + y\nabla x \]

\[ \tilde{x}_1 \ast \tilde{x}_2 = \left[ x_1 \ast x_2, \left( x_1 \ast \left( \frac{\partial \tilde{x}_2}{\partial x_1}, \frac{\partial \tilde{x}_2}{\partial x_2} \right) + x_2 \ast \left( \frac{\partial \tilde{x}_1}{\partial x_1}, \frac{\partial \tilde{x}_1}{\partial x_2} \right) \right) \right] \]
Example of Automatic Differentiation

\[ f(x_1, x_2) = \cos(x_1) + x_1 \cdot \exp(x_2) \]

Procedures:

\[ \bar{x}_1 = [x_1, (1, 0)] \quad \bar{x}_2 = [x_2, (0, 1)] \]

\[ \begin{align*}
\cos(\bar{x}_1) &= \{ \cos(x_1), \quad [-\sin(x_1), 0] \} \\
\exp(\bar{x}_2) &= \{ \exp(x_2), \quad [0, \quad \exp(x_2)] \} \\
\bar{x}_1 \cdot \exp(\bar{x}_2) &= \{ x_1 \exp(x_2), \quad [\exp(x_2), \quad x_1 \exp(x_2)] \} \\
\end{align*} \]

\[ f(x_1, x_2) = \{ \cos(x_1) + x_1 \exp(x_2), \quad [-\sin x_1 + \exp(x_2), \quad x_1 \exp(x_2)] \} \]

\[ \frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \]
Popular AD Packages

• There are many AD packages available: www.autodiff.org
  – OpenAD
  – ADOL-C

• Various implementation methods
  – Source Transformation
  – Operator Overloading

• Various implementation language
  – Fortran
  – C/C++
  – MATLAB
  – Python

• Applications that use AD:
  – Ocean circulation
  – Optimization of dynamic systems governed by PDEs
  – Nuclear reactor applications
Challenges for AD in Reservoir Simulation

1. Variable sparsity pattern

- Diagonal
- Block diagonal
- Stenciling

Evaluation
Challenges for AD in Reservoir Simulation

2. Variable switching
Automatically Differentiable Expression Templates Library (ADETL)

- Developed by Rami Younis at Stanford University
- Initial prototype to prove viability of AD for reservoir simulation
- ADETL solves some reservoir simulation challenges:
  - Block sparse gradient data structure
  - Two algorithms to compute derivative operations
  - Variable switching and adaptive implicit problems
  - Builds runtime expression for derivatives
Sparse Algorithms in ADETL

Algorithm 1: axpy

Phase 1

Phase 2

Zero Initialized Dense Buffer

\[ Y = X_1 + X_2 \]
Sparse Algorithms in ADETL

Algorithm 2: running pointers

\[ a = (7, 0, 3) \]
\[
\begin{array}{c|c|c}
    & 1 & 3 \\
\hline
\text{Col.} & 7 & 3 \\
\text{Value} &   &   
\end{array}
\]

\[ b = (2, 0, 6) \]
\[
\begin{array}{c|c|c}
    & 1 & 3 \\
\hline
\text{Col.} & 2 & 6 \\
\text{Value} &   &   
\end{array}
\]

\[ c = (0, 4, 0) \]
\[
\begin{array}{c|c|c}
    & 2 & 4 \\
\hline
\text{Col.} & 3 & 4 \\
\text{Value} &   &   
\end{array}
\]

\[ d = (0, 0, 4) \]
\[
\begin{array}{c|c|c}
    & 1 & 2 & 3 \\
\hline
\text{Col.} & 9 & 4 & 13 \\
\text{Value} &   &   &   
\end{array}
\]

\[ w = (9, 4, 13) \]
Variable Switching in ADETL

Independent variable set:

Components in liquid phase
[ Po, Pg, Pw, So, Sg, x1, x2, x3, x4, y1, y2, y3, y4]

Components in gas phase
[ Po, Pg, Pw, So, Sg, x1, x2, x3, x4, y1, y2, y3, y4]

Gas phase disappears:

Auto (de)activation
Application of ADETL

• ADETL has been successfully applied in the development of reservoir simulators:
  – AD-GPRS (Stanford University)
  – Unconventional shale gas reservoir simulator (FuRSST)

• We are improving ADETL towards ADETL 2.0
So what’s next?

1. Can we deal with various degrees of sparsity patterns?
2. Can we avoid the cost of runtime sparsity detection?
Non-zero Call Stack

\[ Flux = \sum \rho(P)K_r(S_w)(P_1 - P_2) \]
Univariate

Dense Multivariate

Sparse Multivariate

ADETL
Stage 1 Univariate terms

Test case: product of sequence

\[ F(x) = \prod_{i=1}^{N} f_i(x) \]

\[ F'(x) = \sum_{j=1}^{N} \left( \frac{df_j(x)}{dx} \prod_{i \neq j} f_i(x) \right) \]

<table>
<thead>
<tr>
<th>Case #</th>
<th># of arguments (N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>
Univariate Test 1

1. Hand differentiation

\[ F(x) = \prod_{i=1}^{N} f_i(x) \]
\[ F'(x) = \sum_{j=1}^{N} \left( \frac{df_j(x)}{dx} \prod_{i \neq j} f_i(x) \right) \]

<table>
<thead>
<tr>
<th>N</th>
<th>Value &amp; Derivative Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[ F(x) = f_1(x) ] [ F'(x) = f'_1(x) ]</td>
</tr>
<tr>
<td>2</td>
<td>[ F(x) = f_1(x) \times f_2(x) ] [ F'(x) = f'_1(x) \times f_2(x) + f_1(x) \times f'_2(x) ]</td>
</tr>
<tr>
<td>3</td>
<td>[ F(x) = f_1(x) \times f_2(x) \times f_3(x) ] [ F'(x) = f'_1(x) \times f_2(x) \times f_3(x) + f_1(x) \times f'_2(x) \times f_3(x) + f_1(x) \times f_2(x) \times f'_3(x) ]</td>
</tr>
</tbody>
</table>
Univariate Test 2

2. ADunivariate (new datatype)

```cpp
struct ADunivariate {
    // member functions...
    // data member
    double value;
    double derivative;
};
```

// Multiplication -----------------------------------------------
ADunivariate operator * (const ADunivariate& ad1, const ADunivariate& ad2) {
    return ADunivariate(ad1.value * ad2.value,
                         ad1.value * ad2.derivative + ad1.derivative * ad2.value);
}

<table>
<thead>
<tr>
<th>N</th>
<th>ADunivariate Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R = a</td>
</tr>
<tr>
<td>2</td>
<td>R = a * b</td>
</tr>
<tr>
<td>3</td>
<td>R = a * b * c</td>
</tr>
</tbody>
</table>
3. ADETL
(block sparse gradient)

<table>
<thead>
<tr>
<th>N</th>
<th>ADETL Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R = a</td>
</tr>
<tr>
<td>2</td>
<td>R = a * b</td>
</tr>
<tr>
<td>3</td>
<td>R = a * b * c</td>
</tr>
</tbody>
</table>

// Example of ADETL
adetl::ADscalar a(1.0), b(2.0);
a.make_independent(0);
b.make_independent(0);
adetl::ADscalar c = a * b;
Univariate Test Result

\[ F(x) = \prod_{i=1}^{N} f_i(x) \]

\[ F'(x) = \sum_{j=1}^{N} \left( \frac{df_j(x)}{dx} \prod_{i \neq j} f_i(x) \right) \]

1. Hand Differentiation
2. ADunivariate
3. ADETL

Number of Arguments

<table>
<thead>
<tr>
<th>N</th>
<th>Hand Differentiation</th>
<th>ADunivariate</th>
<th>ADETL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.8X</td>
<td>33X</td>
<td>124X</td>
</tr>
<tr>
<td>4</td>
<td>1.8X</td>
<td>82X</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1.8X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Stage 2 Dense Multivariate

Test case: summation

\[ F(x_1, x_2 \ldots x_n) = \sum_{i=1}^{4} f_i(x_1, x_2 \ldots x_n) \]

\[ \frac{\partial F(x_1, x_2 \ldots x_n)}{\partial x_1} = \sum_{i=1}^{4} \frac{\partial f_i(x_1, x_2 \ldots x_n)}{\partial x_1} \]

\[ \frac{\partial F(x_1, x_2 \ldots x_n)}{\partial x_n} = \sum_{i=1}^{4} \frac{\partial f_i(x_1, x_2 \ldots x_n)}{\partial x_n} \]

<table>
<thead>
<tr>
<th>Case #</th>
<th># of independent variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>80</td>
</tr>
</tbody>
</table>
Dense Multivariate Test 1

1. Manual implementation

```cpp
// Manual Implementation of Dense Vector Addition
for (std::size_t i = 0; i < N; ++i)
{
    Sum[i] = Vector1[i] + Vector2[i]
             + Vector3[i] + Vector4[i];
}
```

Case 1

```
1 2 3 4 5
```

Case 2

```
1 2 3 4 5 ... ... 20
```

Case 3

```
1 2 3 4 5 ... ... ... ... ... 79 80
```
Dense Multivariate Test 2

2. Expression Templates with dense gradient

```c
// Expression Template for Dense Vector Addition
Sum = Vector1 + Vector2
    + Vector3 + Vector4;
```

![Diagram showing vector addition]
3. ADETL (block sparse gradient)

```c
// Example of ADETL for Dense Vector Addition
adetl::ADscalar a(1.0), b(2.0), c(3.0), d(4.0);
// declare independent variables...
adetl::ADscalar Sum = a + b + c + d;
```

<table>
<thead>
<tr>
<th>Case 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Case 3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Problem 2 - Dense Multivariate

\[ F(x_1, x_2 \ldots x_n) = \sum_{i=1}^{N} f_i(x_1, x_2 \ldots x_n) \]
Avoiding sparse operations

sparse  dense

Sparse Jacobian  Seed matrix  Compressed Jacobian
Compressed AD

1. Variable activation
   (Dense Jacobian)

\[ \tilde{U} = \begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1
\end{bmatrix} \]

2. Evaluation

\[ \tilde{J} = \begin{bmatrix}
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times \\
\times & \times
\end{bmatrix} \]

recover

\[ J = \begin{bmatrix}
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times \\
\times & \times & \times
\end{bmatrix} \]
# Challenge with Compressed AD

1. **Sparsity pattern is unknown**
2. **Variable switching**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

- **Sparse Jacobian**
- **Seed matrix**
- **Compressed Jacobian**

\[ x \times \]
ADETL 2.0

- Full range of data types and efficient algorithms
- Ability to convert from one type to the other
- Compressed AD
- Automatic variable set
- Seed matrix will be incorporated
- Benchmark kernel to test AD packages for reservoir simulation calculations
Automatic Differentiation
Algorithms and Data Structures

Chen Fang
PhD Candidate
Joined: January 2013
FuRSST Inaugural Annual Meeting
May 8th, 2014