Dual Number Automatic Differentiation in \textit{RELAP5-3D}

Joshua D. Hodson, Ph.D. Candidate
Utah State University
August 6, 2015
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- *DNAD* – A Fortran implementation of Dual Number Automatic Differentiation
- Integration with *polate*
- Integration with *RELAP5-3D*
- Conclusion
Outline

.gradient Calculation Methods – A Brief Overview

- The Theory of Dual Numbers

- DNAD – A Fortran implementation of Dual Number Automatic Differentiation

- Integration with polate

- Integration with RELAP5-3D

- Conclusion
Gradient Calculation Methods

Modern engineering design & analysis processes rely heavily on computational methods.

Many of these processes require gradient calculations as part of the solution:

- Uncertainty analyses
- Parameter optimization

“... the calculation of gradients is often the most costly step in the optimization cycle...”

Gradient Calculation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Exact Analytical Solutions</th>
<th>Design of Experiments + Finite Difference Approximations</th>
<th>Adjoint Methods</th>
<th>Complex Step Derivatives</th>
<th>Dual Number Automatic Differentiation (DNAD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to implement in existing codes</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Accurate to machine precision</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Efficient run-time performance</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Considers sensitivity of a parameter to multiple inputs</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>High level of maturity</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- DNAD – A Fortran implementation of Dual Number Automatic Differentiation
- Integration with polate
- Integration with RELAP5-3D
- Conclusion
The Theory of Dual Numbers

Start with a Taylor Series expansion about \( x \) by an arbitrary perturbation parameter \( d \):

\[
f(x + d) = f(x) + df'(x) + \frac{d^2}{2} f''(x) + \cdots
\]

Note that the exponent of \( d \) is the same as the order of the derivative for each term in the Taylor Series.

What happens when we multiply two functions together?

\[
h(x + d) = f(x + d) \cdot g(x + d)
\]

\[
= \left( f + df' + \frac{d^2}{2} f'' + \cdots \right) \cdot \left( g + dg' + \frac{d^2}{2} g'' + \cdots \right)
\]

\[
= (fg) + d(fg' + f'g) + \frac{d^2}{2} (fg'' + 2f'g' + f''g) + \cdots
\]
The Theory of Dual Numbers

- This works for every differentiable mathematical operation
- The chain rule of differentiation allows us to string multiple operations together
- This provides the exact analytical equation for derivatives of any order
- A Dual Number is a representation of the first two terms in the Taylor Series (i.e. the function value and its first derivative):
  - \( f(x), f'(x) \)
  - These are exact values, not truncated approximations!
The Theory of Dual Numbers

functions typically have more than one independent variable

For convenience, we can expand the definition of a Dual Number to contain the full gradient of the function:

\[
\begin{pmatrix}
f(x, y, z) \\
fx(x, y, z) \\
fy(x, y, z) \\
fz(x, y, z)
\end{pmatrix}
\]

where subscripts indicate partial derivatives (e.g. \( f_x = \frac{\partial f}{\partial x} \))
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- DNAD – A Fortran module implementation of Dual Number Automatic Differentiation
- Integration with polate
- Integration with RELAP5-3D
- Conclusion
DNAD

DNAD was developed by Dr. Wenbin Yu* as an open-source tool for design optimization

- Minor modifications to make the module more general
  - Number of design variables (independent parameters) can now be specified through a precompiler directive
  - Floating-point precision can now be specified through a precompiler directive
  - Support was added for additional mathematical operations that were not supported in Dr. Yu’s original release

“Simple” process for integrating DNAD into an existing software package:

1. Insert the statement “use dnadmod” at the beginning of each module, function, and subroutine that contain declarations of real numbers
2. Convert all real number declarations to dual number declarations e.g. “real :: x” becomes “type(dual) :: x”
3. Change I/O commands that used to read/write real number data such that the formatting accounts for the extra data contained in a dual number
4. Compile the source code and the DNAD module into a new executable

Use precompiler directives to activate/deactivate DNAD integration at compile-time, so that normal and DNAD executables can be compiled from a single code base
program CircleArea

  real*8, parameter ::
    pi=3.141592653589793d0
  real*8 :: radius, area

  write(*,*) "Enter a radius: ">
  read(*,*) radius
  area = pi * radius**2
  write(*,*) "Area = ", area
end program CircleArea

Modified CircleArea.F90

program CircleArea

  #ifdef dnad
    use dnadmod
    #define real_type type(dual)
  #else
    #define real_type real*8
  #endif

  real_type, parameter ::
    pi=3.141592653589793d0
  real_type :: radius, area

  write(*,*) "Enter a radius: ">
  read(*,*) radius
  area = pi * radius**2
  write(*,*) "Area = ", area
end program CircleArea
**DNAD**

### Original

- **Compiler commands:**
  
  ```
  $ ifort -c CircleArea.F90
  $ ifort CircleArea.F90
  ```

- **Program execution:**
  
  ```
  $ ./a.out
  Enter a radius:
  4.0
  Area = 50.2654824574367
  ```

### Modified

- **Compiler commands:**
  
  ```
  $ ifort -c dnadmod.F90 -Dndv=1
  $ ifort CircleArea.F90 dnadmod.o -Ddnad
  ```

- **Program execution:**
  
  ```
  $ ./a.out
  Enter a radius:
  4.0 1.0
  Area = 50.2654824574367 25.1327412287183
  ```
Previous efforts have been successful in integrating DNAD with existing analysis tools. In each case,

- The code base was relatively small and developed by a single individual
- I/O functions were performed using free formatting
- Only minor modifications were needed to integrate the DNAD module into the software
- DNAD integration proved to be an effective method for accurately and efficiently calculating variable gradients

How does DNAD do with more complex software?
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- DNAD – A Fortran implementation of Dual Number Automatic Differentiation
- Integration with polate
- Integration with RELAP5-3D
- Conclusion
Integration with polate

- polate is a RELAP5-3D-based utility used to calculate fluid properties
- Two state variables and a thermodynamic property table are inputs to the executable
- A complete set of thermodynamic properties needed by RELAP5-3D are output by the executable
- The thermodynamic properties are calculated using various methods
  - Interpolation from thermodynamic property tables
  - Analytical solutions
  - Empirical correlations
Integration with *polate*

- Two of the properties calculated by *polate* are functions of the specific volume ($\nu$):
  - Thermal coefficient of expansion, $\beta = \frac{1}{\nu} \left( \frac{\partial \nu}{\partial T} \right)_P$
  - Coefficient of isothermal compressibility, $\kappa = -\frac{1}{\nu} \left( \frac{\partial \nu}{\partial P} \right)_T$

- *polate* currently uses analytical formulas to evaluate the partial derivatives of specific volume appearing in these equations.

- Finite difference calculations are also used to verify analytical formulas.

- What are the advantages/disadvantages to using *DNAD* for these calculations instead?
Approach:

- Isolate functions used in calculating partial derivatives, and restrict DNAD module to this layer.
- All I/O functions are performed at a higher level and remain unaffected by DNAD integration.
- Use precompiler directives to enable/disable DNAD module and other code changes associated with DNAD integration.
- Convert variable declarations from real to real_type for variables involved in calculating partial derivatives.
- Add timers around functions used in calculating partial derivatives to compare DNAD efficiency with original.
- Calculate percent deviation of dnad-calculated derivatives from derivatives calculated using the analytical solutions.
Integration with *polate*

**Results:**
- Took a considerable amount of effort to integrate *DNAD* module into *polate* (~47 hours)
- *DNAD* version of *polate* ran about 4x slower than unmodified version
- *DNAD* version results were in error by ~0.1-1.0%
  - Error is due to approximations in the modeling algorithm
  - *DNAD* results are exact for the equations being modeled
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- DNAD – A Fortran implementation of Dual Number Automatic Differentiation
- Integration with polate
- Integration with RELAP5-3D
- Conclusion
While it would be nice to be able to accept results from RELAP5-3D as “the answer”, reality is that all engineering analyses have an inherent level of uncertainty associated with their results due to:

- Uncertainties in input parameters
- Uncertainties in the physical models
- Uncertainties in the modeling algorithms

Quantifying the uncertainty in analysis results is crucial to understanding the results
Consider a function \( f \) that is a function of \( n \) independent variables \( x_1 \) through \( x_n \):
\[
f = f(x_1, x_2, \ldots, x_n)
\]

The uncertainty in \( f \) due to input parameters is:
\[
U_f = \sqrt{\sum_{i=1}^{n} \left( U_{x_i} \frac{\partial f}{\partial x_i} \right)^2}
\]
This equation includes the gradient of our function. How do we calculate the gradient?

- Using finite difference methods would require at least 2 separate analyses for each independent parameter, just to get a 1st-order approximation
- Using DNAD, we can get the entire gradient from a single analysis, accurate to machine precision

Limitations of this approach:
- Need to know uncertainty of each input parameter
- Neglects uncertainties in physical models and modeling algorithms
Integration with *RELAP5-3D*

**Current Status:**

- The *RELAP5-3D* installation files have been modified to include the *DNAD* module during compilation.
- The *RELAP5-3D* source files have been automatically modified using a Linux shell script.
- Some source files needed additional modifications due to:
  - A multitude of ways to declare real variables in Fortran
  - equivalence statements
  - data statements that initialize real, integer, and character variables
  - common blocks
Integration with *RELAP5-3D*

- **Remaining work:**
  - Continue addressing source code issues that the shell script was not able to resolve automatically (91 files remain out of 670)
  - **I/O Modifications**
    - Add input routines to allow users to specify design variables through a regular *RELAP5-3D* input deck
    - Modify output file formats to include variable derivatives
  - Compile a new *RELAP5-3D* executable and run validation and benchmarking test cases
  - Complete an uncertainty analysis using the modified *RELAP5-3D* executable and compare to expected results
  - Can this method provide accurate and efficient calculations of variable gradients for use in uncertainty analyses?
  - Are the end results worth the effort of integrating the *DNAD* module with *RELAP5-3D*?
Outline

- Gradient Calculation Methods – A Brief Overview
- The Theory of Dual Numbers
- DNAD – A Fortran implementation of Dual Number Automatic Differentiation
- Integration with polate
- Integration with RELAP5-3D
- Conclusion
Conclusion

Integration of **DNAD** into *polate* offered no benefits over the original method used for calculating derivatives

- Increased runtime and decreased accuracy
- Demonstrated the feasibility of using compiler directives to turn **DNAD** module on or off at compile-time

**DNAD** integration in *RELAP5-3D* has presented unique challenges compared to other investigations due to:

- Large code base (about 760 source files, 300k lines of code)
- Variations in programming styles among the many developers that have contributed to the software

Things to watch out for when integrating **DNAD** into existing software:

- equivalence statements
- data statements
- common blocks
Conclusion

Acknowledgments:

- Dr. Robert Spall, Department Head Mechanical and Aerospace Engineering Utah State University

- Hope Forsmann and the rest of the RELAP5-3D team at INL – thanks for all your help and support this summer!

- Funding for this research is provided by the DOE Office of Nuclear Energy’s Nuclear Energy University Program (NEUP)
Backup Slides
## Gradient Calculation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| Exact Analytical Solutions    | • Accurate to machine precision  
• Simple programming model  
• Efficient run-time performance | • Exact solutions are not always readily available  
• Parameters of interest must be hard-coded into the software |
| Design of Experiments + Finite Difference Approximations | • No modifications to original source code  
• Sensitivity to multiple parameters simultaneously | • Truncation error  
• Sensitive to parameter perturbation sizes  
• Requires an excessive number of simulations |
| Adjoint Methods               | • Accurate to machine precision  
• Sensitivity to multiple parameters simultaneously  
• Fast convergence to optimal design values | • Extensive code modifications required  
• Parameters of interest must be hard-coded into the software |
| Complex Step Derivatives      | • Only requires minor changes to the original source code  
• Parameter of interest is an input option, not hard-coded | • Requires access to source code for initial implementation  
• Truncation error (2\textsuperscript{nd} order)  
• Sensitivity to only one parameter per analysis |
| Dual Number Automatic Differentiation (DNAD) | • Accurate to machine precision  
• Sensitivity to multiple parameters simultaneously  
• Only requires minor changes to the original source code  
• Parameters of interest are input options, not hard-coded | • Requires access to source code for initial implementation  
• Low maturity, limited validation studies |
Defines a derived data type for representing dual numbers:

```fortran
  type, public :: dual
    sequence
    real :: x  ! Functional value
    real :: dx(ndv)  ! Partial derivatives
  end type dual
```

Uses operator overloading to define dual number operations:

```fortran
  public operator (*)
  interface operator (*)
    module procedure mult_dd
  end interface

  elemental function mult_dd(u, v) result(res)
    type(dual), intent(in) :: u, v
    type(dual) :: res
    res%x = u%x * v%x
    res%dx = u%x * v%dx + u%dx * v%x
  end function mult
```
Let’s say we want to know the uncertainty in the Peak Cladding Temperature ($U_{PCT}$) reported by an analysis.

Assume that inputs to the analysis are steady-state power ($P$), thermal conductivity of the gap ($k$), heat capacity of the fuel ($c_p$), and heat transfer coefficient ($h$). (In reality, there could be many more.)

The uncertainty in $PCT$ can then be quantified by the following relationship:

$$U_{PCT} = \sqrt{\left(U_P \frac{\partial PCT}{\partial P}\right)^2 + \left(U_k \frac{\partial PCT}{\partial k}\right)^2 + \left(U_{c_p} \frac{\partial PCT}{\partial c_p}\right)^2 + \left(U_h \frac{\partial PCT}{\partial h}\right)^2}$$