AD-Suite: A Test Suite for Algorithmic Differentiation

M. Narayanamurthi\textsuperscript{1}  T. Bosse\textsuperscript{2}  S. H. K. Narayanan\textsuperscript{2}  P. Hovland\textsuperscript{2}

\textsuperscript{1}Computational Science Laboratory, Department of Computer Science, Virginia Tech
\textsuperscript{2}Mathematics and Computer Science Division, Argonne National Laboratory
What is the need for a test-suite?

- Several AD tools are already available of which many are continually being developed - Tapenade, OpenAD, ADOL-C, ADIC, CppAD, ...
- Simultaneous advancements are being made on both the tool front and their application to larger, more complex and diverse code-bases.

### Advances in tool and theory

- AD software is evolving
  - Checkpointing schemes
  - Interoperating AD-tools
- Theoretical advancements
  - Piece-wise linear and other non-smooth extensions...

### Advances in application

- New computing paradigms
  - Machine-learning
  - Parallel-programming
- New areas of application
- Increasing complexity of current simulation models

### Existing AD tools

[Link to autodiff.org]
What are the challenges facing the development of such a test-suite?

- Existing code samples are authored by different people
- Different tools require code to be in different formats
- Code samples are not available in all major languages
- No system for classifying the examples

Goal

Create a standard AD test-suite that allows scientific validation/comparison with instructive examples that also help new/inexperienced users to apply the different AD-tools.
How do we solve the problem?

- We defined an extensible structure (Folders, Makefiles, READMEs) for the examples.
- We standardized the interfaces of the entry-point functions – for example, interfaces of functions in the driver file.
- We standardized input and output mechanisms in the examples in order to automate the extraction of benchmark data and results for cross-validation.
- We defined a classification system to classify the examples to help users of the test-suite to choose an example depending on their testing needs.
- We built a website to collect these examples and to automatically enforce some of the specifications that we recommend. Additionally, we designed a workflow to handle some aspects of adding a new example to the collection that cannot be reliably automated.
- We collected a few examples that are non-trivial and organized them according to the specifications that we propose.

Disclaimer

The ideas proposed here-in are merely recommendations for the organization of the test-suite. Suggestions on how to improve aspects of the test-suite and needed changes are welcome!
The porous media model is a modified version of that appearing in [Aarnes 2007]. It is a finite volume code that simulates the fracking process in a 3D reservoir. It is a time varying problem which solves a non-linearly coupled system of pressure and saturation equations. The pressure equation is nearly elliptic in nature while the saturation equation is nearly hyperbolic. The porosity and the permeability describe the physical characteristics of the reservoir.

Sequential-splitting method is used to solve this coupled system, where saturation from previous timestep is used to compute saturation dependent quantities, next the pressure equation is solved followed by the velocities. Keeping the velocity constant, the saturation equation is advanced in time and this process is repeated.

The pressure equation is discretised using a two-point flux approximation scheme and an implicit scheme (Newton-Raphson) is used to solve the saturation equation. Sub-timestepping is used to handle convergence issues in the Newton-Raphson method.

The data for the permeability and porosity comes from the SPE-10 dataset. The size of the reservoir is 60 x 220 x 85, making the dimension of problem $\approx 10^6$.

Currently available in MATLAB and Fortran; adjoint and tangent linear version obtained using Tapenade and verified that they agree with each other.
Flow evolution in porous media

- A modified version of that appearing in [Aarnes 2007]
- A time-varying finite-volume code of a 3D reservoir model with non-linearly coupled system of pressure and saturation equations.
- Currently available in MATLAB and Fortran; adjoint and tangent linear version obtained using Tapenade and verified that they agree with each other.

The following graphic illustrates the output from a 2-layer reservoir simulated using (an older version of) the MATLAB code.
Folder structure of porous media problem (and of every other example)

- Code is organized in top-level directories corresponding to each target language.
- Each such directory has a sub-folder named `original_code` and one for each tool that the example has been differentiated with.
- There are Makefiles at each level that contains code at the same level or at a lower level. `make.config` has variable declarations.
- Each example is accompanied with data, documentation and computed results.
Interfaces and specifications for driver files

- Driver file for the original code lives right inside the `original_code` sub-directory. Likewise, those for the differentiated codes can be found under corresponding tool directories.
- The accompanying source files in each case can be found in the `src` sub-directory.

```
program driver
  .
  .
  call get_filepaths(data_directory, results_directory)
call get_independent_size( n_dim, data_directory )
call get_dependent_size( m_dim, data_directory )
call get_parameter_size( p_dim, data_directory )
call allocate_independent_variables( n_dim, x )
call allocate_dependent_variables( m_dim, y )
call allocate_parameter_variables( p_dim, param )
call initialize_independent_variables( n_dim, x, data_directory )
call initialize_dependent_variables( m_dim, y, data_directory )
call initialize_parameter_variables( p_dim, param, data_directory )
call evaluate_original_code(n_dim, m_dim, p_dim, x, y, param, & data_directory, results_directory)
call save_independent_variables( m_dim, y, data_directory )
call deallocate_independent_variables( n_dim, x )
call deallocate_dependent_variables( m_dim, y )
call deallocate_parameter_variables( p_dim, param )
contains
  .
  .
end program driver
```

```
program driver
  .
  .
  call get_filepaths(data_directory, results_directory, dir_x_file)
call get_independent_size( n_dim, data_directory )
call get_dependent_size( m_dim, data_directory )
call get_parameter_size( p_dim, data_directory )
call allocate_independent_variables( n_dim, x )
call allocate_dependent_variables( m_dim, y )
call allocate_parameter_variables( p_dim, param )
call allocate_direction_variables( n_dim, dir_x )
call allocate_derivative_variables( m_dim, deriv_y )
call allocate_independent_variables( n_dim, dir_x_file )
call allocate_direction_variables( n_dim, dir_x, dir_x_file )
call evaluate_deriv_1_dir(n_dim, m_dim, p_dim, x, dir_x, y, deriv_y, param, & data_directory, results_directory, dir_x_file)
call save_dependent_variables( m_dim, y, data_directory )
call save_derivative_variables(m_dim, deriv_y , data_directory )
call deallocate_independent_variables( n_dim, x )
call deallocate_dependent_variables( m_dim, y )
call deallocate_parameter_variables( p_dim, param )
call deallocate_direction_variables( n_dim, dir_x )
call deallocate_derivative_variables( m_dim, deriv_y )
contains
  .
  .
end program driver
```
## Codes for classification of examples

<table>
<thead>
<tr>
<th>Category</th>
<th>Code features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothness of $F$</td>
<td>Lipschitz continuous (C0), $C^1$ (C1), $C^2$ (C2), ..., $C^\infty$ (C∞)</td>
</tr>
<tr>
<td>Source Code Features</td>
<td>Loop (L), (non-)linear Solver (S), Cross-derivative (C), Nested Derivative (N), ...</td>
</tr>
<tr>
<td>Nonlinearity of $F$</td>
<td>Linear (L), Quadratic (Q), Rational (R), Nonlinear (N)</td>
</tr>
<tr>
<td>Sparsity</td>
<td>Sparse (S), Dense (D), Block-structure (B)</td>
</tr>
<tr>
<td>Parallel</td>
<td>Serial (S), Parallel (P),</td>
</tr>
<tr>
<td>Dimensions</td>
<td>n-m-p: Independents (n), Dependents (m), Parameters (p)</td>
</tr>
<tr>
<td>(Special Feature)</td>
<td>(user-specified)</td>
</tr>
</tbody>
</table>

**Table:** Classification codes for different AD applications

### Classification code for porous media model

<table>
<thead>
<tr>
<th>Smoothness</th>
<th>Code Feature</th>
<th>Non-linearity</th>
<th>Sparsity</th>
<th>Parallel</th>
<th>*</th>
<th>Dimensions</th>
<th>Special</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>LS</td>
<td>N</td>
<td>S</td>
<td>S</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Independence, Dependents, Parameters</th>
</tr>
</thead>
</table>

### Feedback on classification codes

- Use levels of smoothness such as “0 = smooth, 1 = piecewise smooth and continuous (via abs, min and max), 2 = Lipschitz (e.g. Euclidean norm), 3 = piecewise smooth discontinuous (e.g. via branching)” instead of $C^k$.

- Recursion (R) could be an additional source-code feature. Also an “extensible and easier to understand front-end needs to be devised”.
**READMEs, Makefiles, data and results directory**

- **README** files should be present in the root directory and in each sub-directory whose contents are not self-explanatory. For instance, if a sub-directory of data directory is material and it contains MATLAB scripts to generate new datasets then an accompanying readme file should explain the procedure to generate new datasets.

- A **Makefile** is present at each level that contains code at the same or at a lower level. Additionally, build flags and variables should be easily configurable in the `make.config` file present in the root level which is included in each Makefile. We have identified a structure (through trial and error) for each Makefile that allows one to easily extend a test problem by adding new versions - in terms of language, tool and mode of differentiation.

- **data** directory contains several sub-folders for each test case — `data_1`, `data_2`, .... **NETCDF** is proposed as format for both writing and reading data. It is understandable, however, that this might not be possible in some scenarios. Files for the input, output and parameters are stored independently with proposed filenames being `x.nc`, `y.nc` and `parametersi.nc`. There can be multiple parameters file such as `parameters1.nc`, `parameters2.nc`, `parameters3.nc` .... All scalar parameters can go into a single file and a separate file for each array parameter can be maintained. Input files for tangent directions can be labeled `dir_x_i.nc` for each separate direction that needs to be evaluated and is passed as command-line argument to the executable at runtime. Likewise, the adjoint directions are read from `adj_y_i.nc`. Output from the tangent-linear model is written to `deriv_y.nc` and that of the adjoint model to `grad_x.nc`

- **results** directory will have a similar structure to the example itself with separate sub-directories for each target language and results will be written to a file name corresponding to the driver that generated it.
Overview of the build process

/include/Makefile

#include make.config

define HELP_BODY

This makefile has the following Options and corresponding values that can be passed to it from the command line while calling make.

<table>
<thead>
<tr>
<th>OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANG = &lt;ALL/F90/MATLAB&gt;</td>
</tr>
<tr>
<td>DERIV = &lt;ALL/TAPENADE/OPENAD&gt;</td>
</tr>
</tbody>
</table>

export HELP_BODY

#Space separated list of folders to build
BUILDS = f90 m

.PHONY: all build help clean

all: build

build: $(BUILDS)
  $(foreach d, $^, make -C $(d) build);

help:
  @echo "$HELP_BODY"

clean: $(BUILDS)
  $(foreach d, $^, make -C $(d) clean);

/f90/Makefile

#include ../make.config

BUILDS = original_code tape

build: $(BUILDS)
  $(foreach d, $^, make -C $(d) build);

clean: $(BUILDS)
  $(foreach d, $^, make -C $(d) clean);

/f90/original_code/src/Makefile

Custom build script

/f90/original_code/Makefile

Custom build and link script

/f90/tapenade/Makefile

#include ../..make.config

...

# Determine if this directory needs to be built
# Refer: http://stackoverflow.com/a/28790801/1220495
IS_F90TRAN=$(shell echo $(LANG)|grep -i "-c F90")
IS_TAPENADE=$(shell echo $(DERIV)|grep -i "-c TAPENADE")
IS_ALL_LANG=$(shell echo $(LANG)|grep -i "-c ALL")
IS_ALL_DERIV=$(shell echo $(DERIV)|grep -i "-c ALL")
FLAG=$(shell echo $$((($IS_F90TRAN) + $(IS_ALL_LANG)) * ($IS_TAPENADE) + $(IS_ALL_DERIV))))

# Add targets if directory needs to be built
CONDITIONAL_BUILD =
  ifeq ($(FLAG), 0)
    CONDITIONAL_BUILD += cdir cproj src $(OBJS_MAIN_TL) $(OBJS_MAIN_ADJ) link
  endif

.PHONY: cdir cproj src build link

build: $(CONDITIONAL_BUILD)

...
We have built a website to collect and organize the test examples, and to make it easy to find examples that suit particular testing requirements.

It is based on Drupal 7 with custom modules to perform some specific tasks written in PHP.

Currently two user-levels are supported – AD-Suite User and AD-Suite Admin.

The website can automatically check whether the directory structure of the uploaded example is in line with the laid specifications.

A workflow such as the one below is needed to complete the chain from submission to making the example accessible to public through the website and public GIT repository.
A test-suite for algorithmic differentiation

AD-Suite is a collection of non-trivial test examples in different areas of applications that have been differentiated with one or more AD tools. It is used as a test-bed by AD developers to test advances in AD tools. The examples can also be used by AD users as a template to learn how to apply AD to their own codes. Examples are categorized by classification codes that describe different features about the code in a concise manner. When testing specific features of an AD tool, these classification codes will prove to be handy in choosing an appropriate example for the test.

Test problems in the AD-Suite are available in almost all major computer programming languages such as C, C++, Fortran 77, 90, Matlab, Java, and Java. An extensible structure with a modular build system makes it easy to contribute new implementations to the test suites. Please visit the documentation link from the menu at the top to learn how you can participate.
Porous Media

Submitted by ADSuite Admin on Wed, 07/27/2016 - 13:55

Porous Media example illustrates the process of fracturing. Derivatives of the total output all may be computed against the input flow which can be used in an optimization setting to increase oil production output.

Languages:
Fortran 90
Java
Python
Matlab
Multiple

AD Tools:
TapeADE

GIT URL:

Authors:
Jang T. Aarnes
Tore Glimm
Knut-Arne Loh
Malalesh Narsarao Murtli
Timor E. Eriksen
Sri Hari Krishna Narayanan
Paul Houle

A test-suite for algorithmic differentiation

AD Suite is a collection of non-trivial test examples in different areas of application that have been differentiated with one or more AD tools. It is used as a test-bed by AD developers to test advances in AD tools. The examples can also be used by AD users as a template to learn how to apply AD to their own codes. Examples are categorized by classification codes that describe different features about the code in a concise manner. When testing specific features of an AD tool, these classification codes will prove to be handy in choosing an appropriate example for the test.

Test problems in the AD Suite are available in almost all major computer programming languages such as C, C++, Fortran 77/90, Python, Matlab, Java and R. An extensible structure with a modular build system make it easy to contribute new implementations to the test-suite. Please visit the documentation link from the menu at the top to learn how you can participate.
Current status of the test-suite and contributions from community

- We have collected close to ten examples of which two of them are (almost) in compliance with the recommended specifications — airfoil and porous media.

- Following applications will be made available through the repository:
  - **airfoil** — based on [Giles 2005]
  - **porous media** — based on [Aarnes 2007]
  - power generation — contributed by Shrirang Abhyankar, Vishwas Rao and Mihai Anitescu
  - 2 atmospheric chemistry models generated using KPP — kintetic pre-processor [Damian 2002]
  - simulated moving bed — contributed by Andrea Walther and Kshitij Kulshreshtha
  - sonic boom — contributed by Laurent Hascoet and Sri Hari Krishna Narayanan
  - streamvel — contributed by Daniel Goldberg and Sri Hari Krishna Narayanan
  - 2D Burger’s equation — contributed by Max Sagebaum and Nico Gauger

- We have also received significant contributions from the following people: Torsten Bosse, Sri Hari Krishna Narayanan, Kshitij Kulshreshtha, Mu Wang and Rijul Rastogi.
Summary

- We started to collect/write a comprehensive test-suite for AD containing several realistic applications for AD-beginner, AD-User, and AD-developer, e.g., porous media example.
- The test suite has clearly defined specifications that include
  - organizational structure,
  - interfaces of entry points into the code
  - a classification system to categorize test-examples.
- The input-output mechanisms were standardized by resorting to the use of NETCDF, although alternate approaches are welcome.
- A modular build system makes the addition of variants of examples tailored for specific languages or AD tools a breeze.
- A web-interface has been built to automate verification of some of the specifications; it also facilitates users to contribute test examples and to download existing examples.

Outlook

- We aim to finish the organization of the examples that we have collected in the proposed format and finalize the website to allow submission and distribution of examples.
- We look forward to participation from the community in contributing test-examples.
- We want to encourage more folks to try AD by showcasing the examples that we collect in the repository.
This work would not have been possible without the continued trust and support of my collaborators at Argonne National Lab - Krishna, Torsten and Paul - and of my advisor, Adrian Sandu, at Virginia Tech.

I thank Argonne National Lab and SIAM for funding my travel to this conference.
References

▶ autodiff.org


