TSOpt 2.0: An Overview

Marco Enriquez

The Rice Inversion Project
marco.enriquez@caam.rice.edu

TRIP Annual Meeting

February 20, 2009
Simulation-Driven Optimization Problems

We are interested in solving optimization problems constrained by differential equations,

$$\min_{c} \quad J(c) = G(u(c, \cdot))$$

$$s.t. \quad \bar{H} \left( \frac{du}{dt}, u, c \right) = 0,$$

given that we have an application package capable of solving the state equation.

Examples:

- Given injector/producer well locations, find well rates that maximize revenue, subject to the black-oil equations
- Seismic Inversion (TRIP afternoon talks)
TSOpt ("Time Stepping For Optimization")

TSOpt is TRIP’s “middle-ware” package. TSOpt:

- abstracts commonalities among time-stepping methods
- provides a way for a simulation package to inter-operate with optimization algorithms

Extra Features:

- implements the Adjoint-State method to form gradients
- allows efficient way to verify reference, derivative and adjoint simulation are appropriately related
TSOpt ("Time Stepping For Optimization")

Diagram:
- Optimizer
- TSOpt
- Simulator

M. Enriquez  TSOpt 2.0: An Overview – 3
TSOpt ("Time Stepping For Optimization")

\[ u^+ = u + \Delta t H(u, c, t) \]
TSOpt ("Time Stepping For Optimization")

\[ \nabla J(c), J(c), \ldots \]

Optimizer  \hspace{2cm} TSOpt  \hspace{2cm} Simulator
TSOpt ("Time Stepping For Optimization")

\[
    s = -B(c)^{-1} \nabla J(c)
\]

\[
    c^+ = c + \alpha s
\]
TSOpt ("Time Stepping For Optimization")

\[ C = C^+ \]
TSOpt and the AS Method

The AS method requires access to the reference simulation state history.

TSOpt implements the following strategies to address this:

- **save all**: save states as you forward simulate, access as needed
  - Cost: \( TB \)s, for a typical 3D RTM.

- **checkpoint**: rely on forward simulations, *and* use stored simulation states as a starting point for evolution
  - Cost: \( O(\log(N)) \) recomputation, given a special distribution of the states and a small amount of buffers
  - Two flavors: offline and online

- **specialized strategies for specific problems**
  - RTM: only save boundary values
TSOpt and The Alg Framework

TSOpt’s components derive from TRIP’s Alg package, a software framework that can be used to describe any algorithm.

The Alg package defines two main objects:
- Algorithm objects, which must implement void run()
- Terminator objects, which must implement bool query()

By using these two objects, we may create a variety of algorithms:
- composite algorithms: `{ alg1.run(); alg2.run() }
- iterative algorithms: while(!term.query()) { alg.run(); }
TSOpt’s Components

In TSOpt, we use Jet objects to perform various simulations. Hence, a Jet object “holds” information on how to take forward, derivative and adjoint evolution steps.
In TSOpt, we use Jet objects to perform various simulations. Hence, a Jet object “holds” information on how to take forward, derivative and adjoint evolution steps.

All of these classes are templated on a State class, which itself holds state data and a time object.
Running Simulations in TSOpt

Typically, this is how we create a 1-jet:

```cpp
FwdTimeStep stp(...); // Forward Evolution
FwdDTimeTerm<State> tt(...); tt.setTargetTime(nt); // Forward Terminator
SaveAllSim<State, containerClass> f(stp, tt); // Save all fwd. states

DerTimeStep dstp(...); // Derivative Evolution
FwdDTimeTerm<State> dtt(..); dtt.setTargetTime(nt); // Derivative Terminator

AdjTimeStep astp(...); // Adjoint Evolution
BwdDTimeTerm<State> att(..); att.setTargetTime(0); // Adjoint Terminator

StdJet<State> j(f, dstp, dtt, astp, att); // Create a jet
j.getAdj().run(); // Run adjoint sim.
```
Running Simulations in TSOpt

To use checkpointing in TSOpt, we only change the following line:

```cpp
FwdTimeStep stp(...); // Forward Evolution
FwdDTimeTerm<State> tt(...); tt.setTargetTime(nt); // Forward Terminator
CPSim<State, containerClass> f(stp, tt, numBuffers); // Checkpoint

derTimeStep dsrc(...); // Derivative Evolution
FwdDTimeTerm<State> dtt(...); dtt.setTargetTime(nt); // Derivative Terminator

AdjTimeStep astp(...); // Adjoint Evolution
BwdDTimeTerm<State> att(...); att.setTargetTime(0); // Adjoint Terminator

StdJet<State> j(f, dsrc, dtt, astp, att); // Create a jet
j.getAdj().run(); // Run adjoint sim.
```
A Unit Test Problem

Consider the following initial value ODE problem:

\[ u_t = 1 - u^2 \]
\[ u(0) = 0.5, \quad t \in [0, 0.1] \]

Let’s perform the adjoint evolution with the following strategies to handle the reference states:

- save all
- checkpoint

and verify results via the *dot product* test.