Adaptive Time Stepping for Optimal Control Problems

Marco Enriquez

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

December 9, 2010

Simulation-Driven Optimization Problems

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

$$\begin{aligned} \min_{c} \qquad & J(c) = G(u(c,\cdot)) \\ s.t. \qquad & \bar{H}\left(\frac{du}{dt}, u, c\right) = 0\,, \end{aligned}$$

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

Other Examples:

- History Matching
- Seismic Inversion (Dong)

Motivating Problem

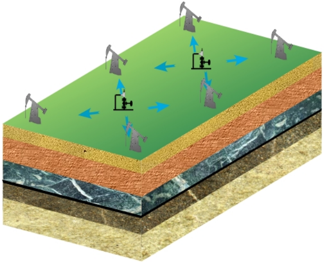
Suppose the following:

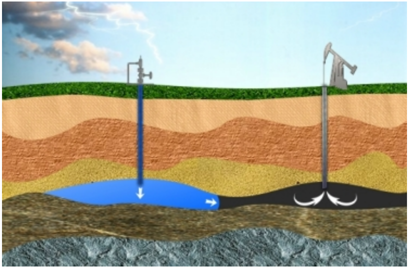
1. We use derivative-based methods to solve [SD], relying on the **adjoint-state method** to obtain derivatives of J
2. The solution of the state equation *changes rapidly* in certain time intervals, motivating use of **adaptive time-stepping**

How will this affect the numerical approach we use to solve [SD]?

Motivating Example: Optimal Well Rate Allocation

[OWRA]: Given a reservoir model, along with location of injection and production wells, find the optimal well rates to maximize revenue





[1]Images courtesy of www.amerexco.com/recovery

Motivating Example: Optimal Well Rate Allocation

$$\max_{q_i \quad i\in I\cup P} \quad J(q) = \int_0^T dt \left(\sum_{i\in P} \alpha(1-s_a)q_i(t) - \sum_{i\in P} \frac{\beta}{2} s_a q_i^2(t) - \sum_{i\in I} \gamma q_i(t) \right) ,$$

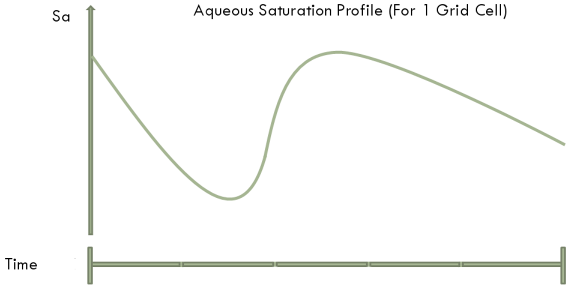
where α, β and γ are scalar variables and the aqueous pressure p and aqueous saturation s_a solve:

$$\begin{aligned} -\nabla \cdot (K(x)\lambda_{tot}(s_a(x,t))\nabla p(x,t)) &= \sum_{i\in P}(1-s_a)q_i(t)\delta(x-x_i) \\ &\quad + \sum_{i\in P\cup I} s_a q_i(t)\delta(x-x_i) \\ \phi(x)\frac{\partial}{\partial t}s_a(x,t) - \nabla \cdot (K(x)\lambda_a(s_a(x,t))\nabla p(x,t)) &= \sum_{i\in P\cup I} s_a q_i(t)\delta(x-x_i) \end{aligned}$$

[1] Problem formulation from Wiegand et al., *Adjoint calculations for a reservoir management problem*

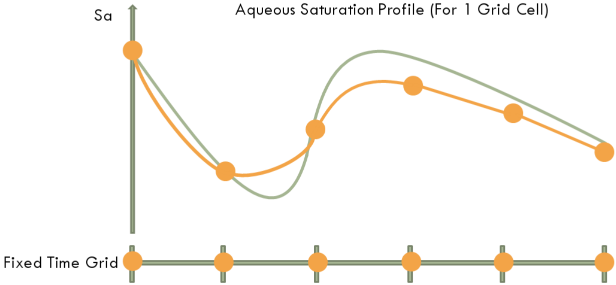
Motivating Example: Optimal Well Rate Allocation

Rapid changes in the wellrates (q) lead to rapid variation in the solution of the Black-Oil Equations



Motivating Example: Optimal Well Rate Allocation

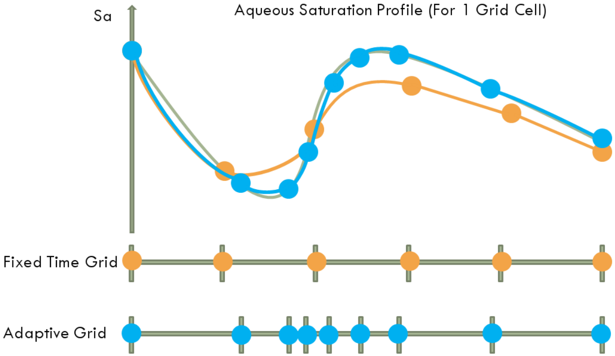
Rapid changes in the wellrates (q) lead to rapid variation in the solution of the Black-Oil Equations



Motivating Example: Optimal Well Rate Allocation

Rapid changes in the wellrates (q) lead to rapid variation in the solution of the Black-Oil Equations

- Adaptive time-stepping is common feature in industrial reservoir simulators



Adaptive Time Stepping

Adaptive time stepping is the preferred method for solving differential equations with rapidly changing solutions

- Requires an input: error tolerance τ
- Steplengths expand or contract, to maintain solution error of $O(\tau)$

How to use adaptive time stepping with the adjoint state method?

- In order to use adaptive time stepping to solve [SD], we apply the optimality conditions to [SD], *before* discretizing

The Continuous Adjoint-State Method

Applying the optimality conditions to [SD], for $t \in [0, T]$:

Continuous State Equation:

$$\frac{du}{dt} = H(u(t), c) \qquad u(0) \equiv 0$$

Continuous Adjoint Equation:

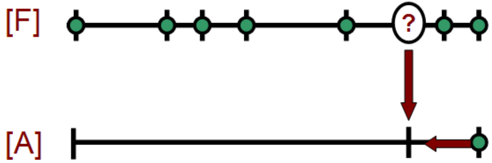
$$\frac{dw}{dt} = -D_u H(u(t), c)^* w(t) + J_u(u(t), c) \qquad w(T) \equiv 0$$

Gradient:

$$\nabla f(c) = \int_0^T D_c H(u(t), c)^* w(t) + J_c(u(t), c) dt$$

Adaptive Time Stepping and the Adjoint State Equations

Solve the state and adjoint equations above via adaptive time-stepping



Problem: Mismatched time grids

- Interpolation is needed to complete the adjoint evolution
- Interpolation Error $\longrightarrow$ Adjoint Error $\longrightarrow$ Gradient Error
- **Claim:** Despite interpolation error, we can still guarantee local convergence to [SD]

The Adaptive Tolerance Method

Claim: Suppose we solve [SD] with the Newton method and use adaptive time-stepping to resolve the DE constraints.

Using the following time-stepping tolerance update:

$$\tau_{k+1} = \min(\tau_k, \|g_k\|^p), \quad p \in (1, 2]$$

is enough to guarantee local convergence to a stationary point

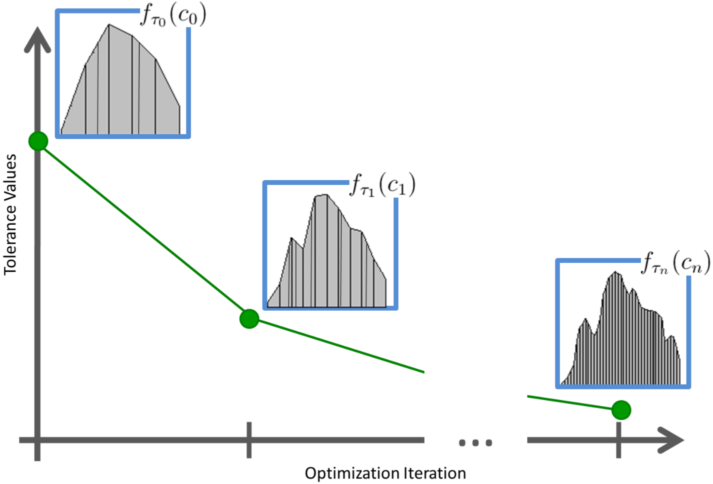
Algorithm: Adaptive Tolerance Method

a. Set initial time-stepper tolerance τ_0 and initial control c_0. Set $k = 0$.

b. `while` (optimization error $< tol_{opt}$)

   1. With τ_k and c_k, solve reference and adjoint equations.
   2. Take Newton Step: solve $H_k s_k = g_k$, then $c_{k+1} = c_k + s_k$.
   3. $\tau_{k+1} = \min(\tau_k, \kappa(\text{optimization error})^p)$ for $p \in (1, 2]$.
   4. Set $k = k + 1$.

The Adaptive Tolerance Method



TSOpt ("Time Stepping For Optimization")

TSOpt is "middle-ware" written in C++, designed to aid solution of simulation-driven optimization problems

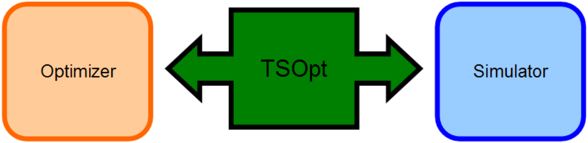
TSOpt:

- abstracts commonalities among time-stepping methods
- provides a way for a simulation package to inter-operate with optimization algorithms
- supports use of the adjoint-state method

Motivating observation: for every simulation driven optimization problem, the solution process is (mostly) the same:

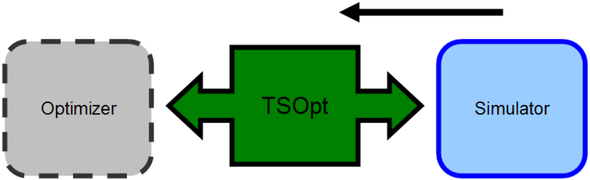
- reference, linearized and adjoint simulation execution order
- constructing needed data structures for optimization

TSOpt ("Time Stepping For Optimization")

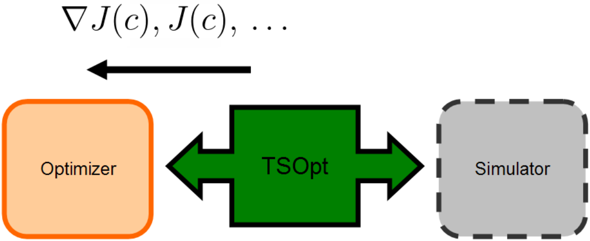


TSOpt ("Time Stepping For Optimization")

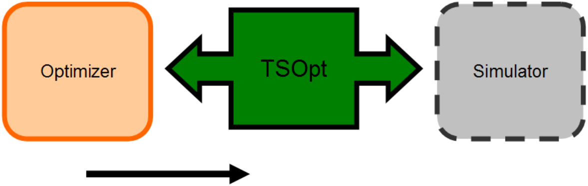
$$u^+ = u + \Delta t H(u, c, t)$$



TSOpt ("Time Stepping For Optimization")



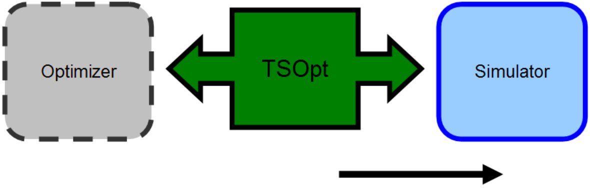
TSOpt ("Time Stepping For Optimization")



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

$$c^{+} = c + \alpha s$$

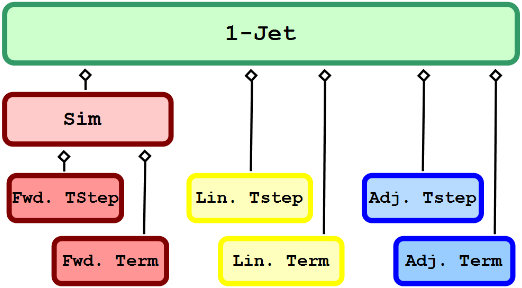
TSOpt ("Time Stepping For Optimization")



$$c = c^{+}$$

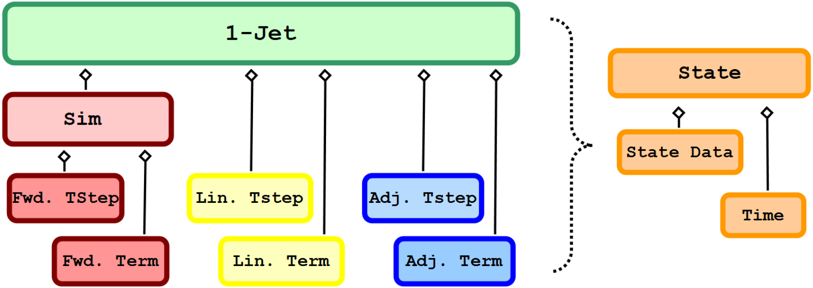
TSOpt's Components

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.



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.



All of these classes are templated on a State class, which itself holds state data and a time object

Inversion Software Construction

A consequence of `TSOpt`'s modular structure is that it minimizes the amount of code needed to perform an inversion

User:

- provides `TSOpt` with a forward, linearized, and adjoint "step"
- provide a "State" class

`TSOpt`:

- arranges proper execution forward, linearized and adjoint simulation
- implements the Adjoint-State method to form gradients

Output can be passed to optimization software

TSOpt and the Adjoint-State (AS) Method

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

TSOpt implements the following strategies, for both fixed-step and adaptive time-stepping:

- **save all**: save states as you forward simulate, access as needed
  - Cost: A typical 3D RTM, $O(\text{TB})$
- **checkpointing**: 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
- **specialized strategies for specific problems**

A Checkpointing Example

Consider a 15 day simulation, with $dt = 1$ day. Checkpoint with 3 buffers.

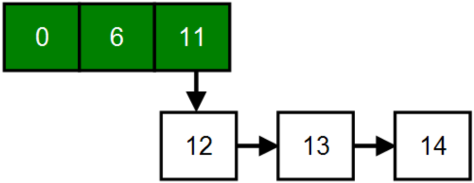
Checkpointing Initial Steps:

1. Figure out which states to save.
2. Run forward simulation.
3. Store states at times $t = 0, 6, 11$ into the 3 buffers.

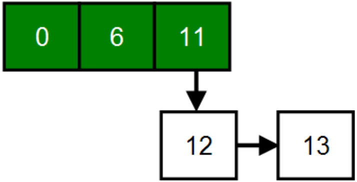
The first adjoint step: solve for the adjoint variable at $t = 14$

- Requires access to simulation state at $t = 14$

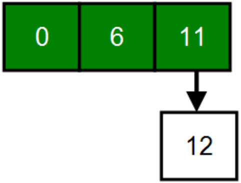
2: From the Last CP, Timestep to Generate u_{14}



3: From the Last CP, Timestep to Generate u_{13}

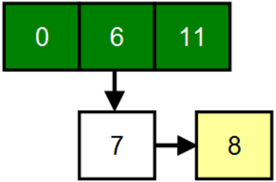


4: From the Last CP, Timestep to Generate u_{12}



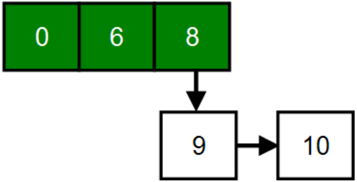
5: Since We Stored It, Access u_{11}

6: From u_6 , Generate New CP

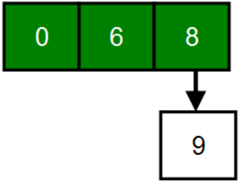


7: Overwrite Useless Buffer with New CP

8: From Updated Last CP, Timestep to Generate u_{10}

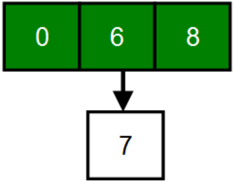


9: From Updated Last CP, Timestep to Generate u_9



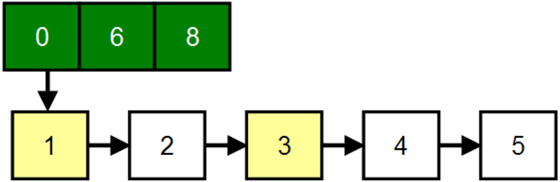
10: Since We Stored It, Access u_8

11: From Second Stored CP, Timestep to Generate u_7

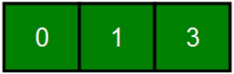


12: Since We Stored It, Access u_6

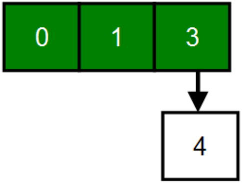
13: From First CP, Timestep to Generate u_5, Gen. 2 CPs



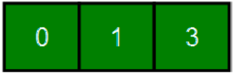
14: Overwrite Buffers with 2 New CPs



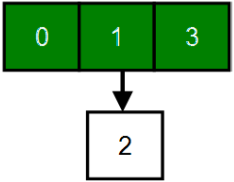
15: From Last CP, Timestep to Generate u_4



16: Since We Stored It, Access u_3



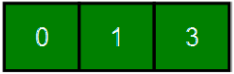
17: From Second CP, Timestep to Generate u_2



18: Since We Stored It, Access u_1

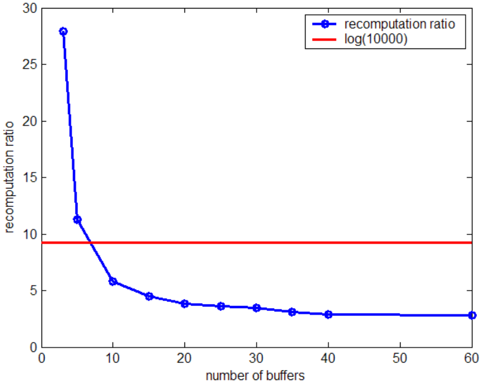
0	1	3

19: Since We Stored It, Access u_0



Recomputation Cost of Checkpointing

Consider the following case, where $N = 10000$



buffers	3	5	10	15	20	25	30	35	40	60
ratio	27.9	11.3	5.8	4.5	3.8	3.6	3.4	3.1	2.9	2.8

Simulation Verification

In order to obtain meaningful results from inversion, one must guarantee that the gradient is accurate

Gradient quality depends on the adjoint states, which depends on:

- linearization of the reference equations
- adjoint of the linearization

`TSOpt` is capable of the following simulation verification (**unit**) tests:

- **derivative test**: compare linearized simulation to finite difference approximation (using reference simulation)
- **dot product test**: give the linearized simulation operator A, adjoint simulation operator A^* and random control x and random state y, check $\langle Ax, y\rangle - \langle x, A^*y\rangle$ (Fixed timestep only)

The Optimal Well Rate Allocation Problem

Recall the optimal well rate allocation problem:

$$\min_{q_i \; i \in I \cup P} \quad J(q) = \int_0^T dt \left(\sum_{i \in P} \alpha(1 - s_a) q_i(t) - \sum_{i \in P} \frac{\beta}{2} s_a q_i^2(t) - \sum_{i \in I} \gamma q_i(t) \right) ,$$

where α, β and γ are scalar variables and the aqueous pressure p and aqueous saturation s_a solve:

$$\begin{aligned} -\nabla \cdot (K(x)\lambda_{tot}(s_a(x,t))\nabla p(x,t)) &= \sum_{i \in P} (1 - s_a) q_i(t) \delta(x - x_i) \\ &\quad + \sum_{i \in P \cup I} s_a q_i(t) \delta(x - x_i) \\ \phi(x) \frac{\partial}{\partial t} s_a(x,t) - \nabla \cdot (K(x)\lambda_a(s_a(x,t))\nabla p(x,t)) &= \sum_{i \in P \cup I} s_a q_i(t) \delta(x - x_i) \end{aligned}$$

Fully Discretized Problem

After using a Finite Volume method in space and a 1-2 scheme in time (Bwd. Euler + Trapezoid Rule):

$$\begin{aligned} \min \quad & \bar{J}(q) = \sum_{k=1}^{N} h^k \, l(t^k, s_a^{(t^k)}, q) \\ \text{s.t.} \quad & e^T q = 0 \\ & q_{min} \leq q_i \leq q_{max} \end{aligned}$$

where $s_a^{(t^{k+1})}$ and $p^{(t^{k+1})}$ solve:

$$\begin{bmatrix} f(\ldots^{(t^{k+1})}, q) \\ g(\ldots^{(t^{k+1})}, q) \end{bmatrix} := \begin{bmatrix} \varphi[q](t^{k+1}) - Ap^{(t^{k+1})} \\ D^{-1}(\varphi[q](t^{k+1}) - \tilde{A}p^{(t_{k+1})}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_a^{(t^{k+1})} - s_a^{(t^k)}}{h^k} \end{bmatrix}$$

where the matrices $A^{(\theta)}$ and D are defined as:

$$\begin{aligned} D_{i,i} &= \phi_i \cdot |\Omega_i| \\ A_{i,j}^{(\theta)} &= -T_{i,j}\lambda_{\theta_{i,j}} \qquad A_{i,i}^{(\theta)} = \sum_j T_{i,j}\lambda_{\theta_{i,j}} \end{aligned}$$

The Adjoint Equations

Simultaneously solve for the adjoint variables $w_s^{(t^k)}$ and $w_p^{(t^k)}$ in the following equation:

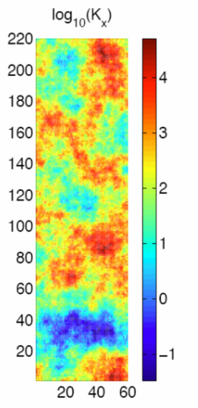
$$\begin{aligned} -\frac{w_s^{(t^{k+1})} - w_s^{(t^k)}}{h^k} &= D_s f(\ldots^{(t^k)})^T w_s^{(t^k)} - D_s g(\ldots^{(t^k)})^T w_p^{(t^k)} - \nabla_s l(\ldots^{(t^k)}) \\ 0 &= -D_p f(\ldots^{(t^k)})^T w_s^{(t^k)} + D_p g(\ldots^{(t^k)})^T w_p^{(t^k)} \end{aligned}$$

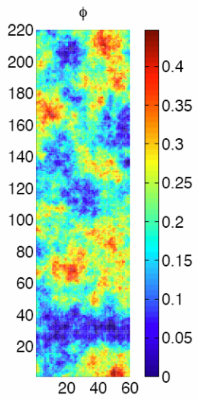
The directional derivative can then be obtained from the following expression:

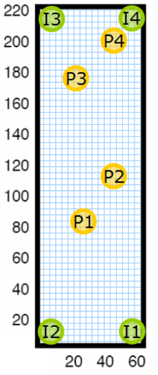
$$\nabla J(q) = \Delta q \sum_{i=1}^{N} \nabla_q l(\cdot^{(i\Delta q)}) - D_q f(\ldots^{(i\Delta q)})^T w_s^{(i\Delta q)} + D_q g(\ldots^{(i\Delta q)})^T w_p^{(i\Delta q)}$$

Simulation Information

- SPE10 data for porosity and permeability (left)
- Location of Injecting/Producing Wells (right)
- Grid Cell Size: 10×20 feet

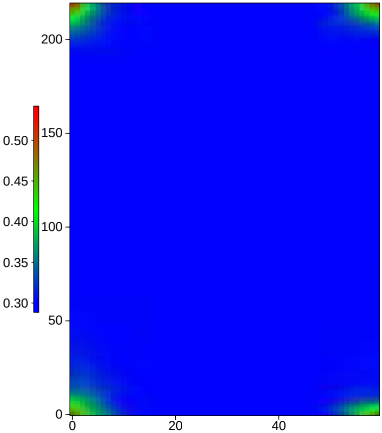






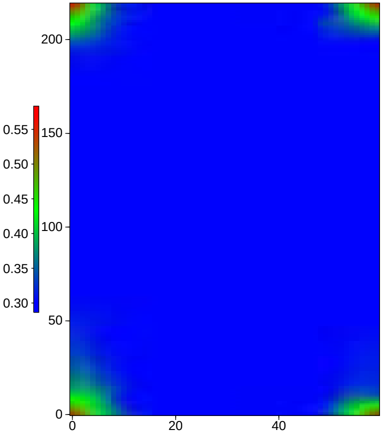
Reference Simulation Results

Saturation plot for $t = 25$ days



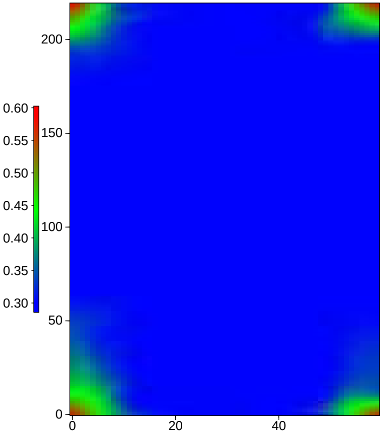
Reference Simulation Results

Saturation plot for $t = 50$ days



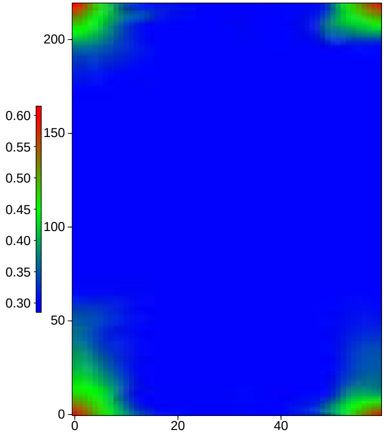
Reference Simulation Results

Saturation plot for $t = 75$ days



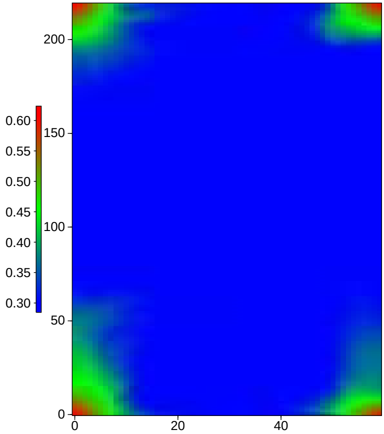
Reference Simulation Results

Saturation plot for $t = 100$ days



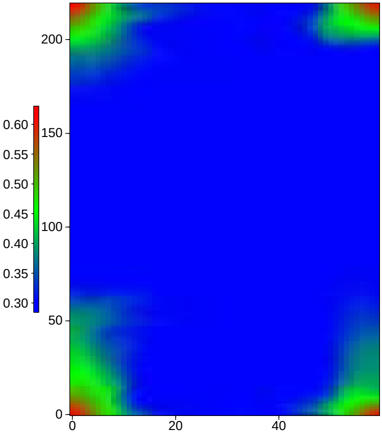
Reference Simulation Results

Saturation plot for $t = 125$ days



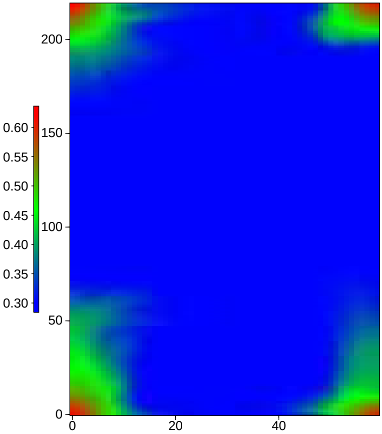
Reference Simulation Results

Saturation plot for $t = 150$ days



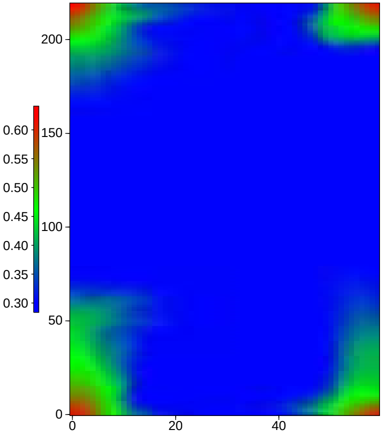
Reference Simulation Results

Saturation plot for $t = 175$ days



Reference Simulation Results

Saturation plot for $t = 200$ days



Inversion Information

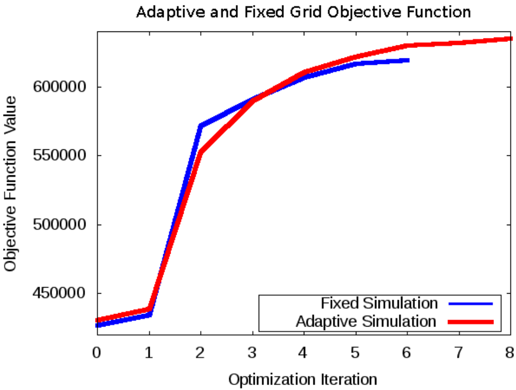
Computational Software:

- Simulation: `BlackOil` simulator
- `TSOpt` to handle simulation execution, gradient construction
- Optimization: `IPOpt`, "Interior-Point Optimizer"

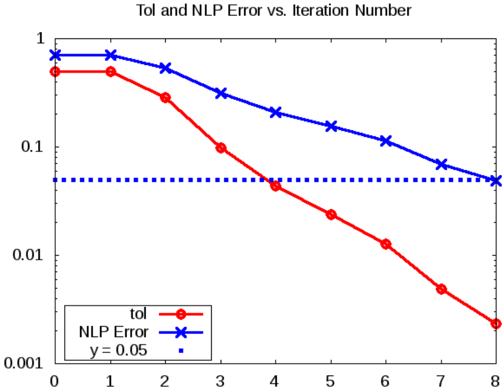
Inversion:

- Find optimal well-rate configuration over 200-day timespan
- Stopping tol.: `5e-2` NLP error
- LBFGS Hessian approximation
- Globalization: Linesearch
- Wellrate bounds: $[0, 20]$ bbl/day
- Initial guess: 10 bbl/day for all wells

Objective Function



NLP Error vs. Tolerance Values



Error vs. Compute-Time Comparison

To reach 11% NLP error:

- Fixed: 9^{+} hrs., $\Delta t = 0.25$
- Adaptive: 3 hrs.

Conclusions

Fixed-step approach to solving optimal control problems with DE constraints with rapidly-varying solutions

- Requires fine time grid for accuracy (**Expensive**)

Adaptive Approach:

- Requires OtD approach
- Higher sim. accuracy $\rightarrow$ accurate derivatives $\rightarrow$ better optim. results
- Adaptive tolerance method: solves DE as accurately as needed

Conclusions

`TSOpt`:

- Modular `C++` framework aiding inversion software construction
- Easily switch between strategies for inversion and gradient formation
- Supports checkpointing for fixed and adaptive simulations

Using the Adaptive Tolerance Method for OWRA:

- Solved via `BlackOil` + `TSOpt` + `IPOpt`
- Increase in projected revenue (3%)
- Reached NLP error of 5%

Questions?

Gradient and Hessian Error

Theorem: Let g and H be the computed gradient and Hessian, respectively. If the reference and adjoint equations are solved adaptively with tolerance τ, then:

$$\begin{aligned} \|g - \nabla f(c)\| &\leq C_g \, \tau \\ \|(H - \nabla^2 f(c)) \, p\| &\leq C_H \, \tau \end{aligned}$$

for constants $C_g, C_H > 0$ and a search direction p.

Inexact Optimization Algorithms

How will the derivative error affect solution of the optimal control problem?

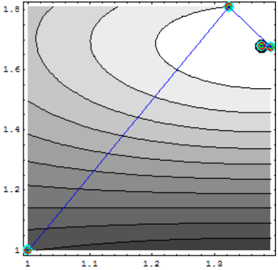
Inexact Optimization Algorithms:

- Theoretically guarantees convergence, despite derivative error
- Focus: Inexact Newton Methods
- **Idea:** Couple derivative error to inexact Newton theory

The Inexact Newton Method

Consider the following problem:

$$\min_c f(c)\,, \qquad f : \mathbb{R}^n \to \mathbb{R}$$



Standard Newton:

Solve: $\nabla^2 f(c)\, s = \nabla f(c)$

Update: $c^+ = c + s$

Inexact Newton Algorithm

Solve $\nabla^2 f(c)\, s = \nabla f(c) + r(c)$

Update: $c^+ = c + s$

- Local convergence if $\|r(c)\| \leq K \cdot \|\nabla f(c)\|^p$ for $p \in (1, 2]$

The Adaptive Tolerance Method

Insight: If the derivative discretization error at the k^{th} iteration,

$$\|r_k\| \approx C\,\tau_k\,,$$

then the inexact Newton criterion

$$\|r_k\| \leq K \cdot \|\nabla f(c_k)\|^p\,, \quad p \in (1,2]$$

yields an **update scheme** for the tolerance

The Adaptive Tolerance Method

Claim: Suppose we solve [SD] with the Newton method and use adaptive time-stepping to resolve the DE constraints.

Using the following time-stepping tolerance update:

$$\tau_{k+1} = \min(\tau_k, \|g_k\|^p), \quad p \in (1, 2]$$

is enough to guarantee local convergence to a stationary point

Adaptive Checkpointing

This algorithm stems from Walter's `ARevolve`:

- **Good:** Recomputation cost close to optimal $(log(N))$, plus small penalty due to adaptivity
- **Bad:** Assumes reference time grid and adjoint time grid align

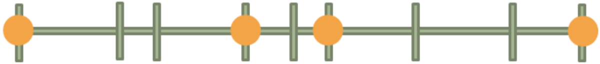
Goal: Keep the near-optimal recomputation ratio, without the restriction on the time grids

Solution:

- Add interpolation buffer that moves with the adjoint evolution
- Manage calls are made to `ARevolve`

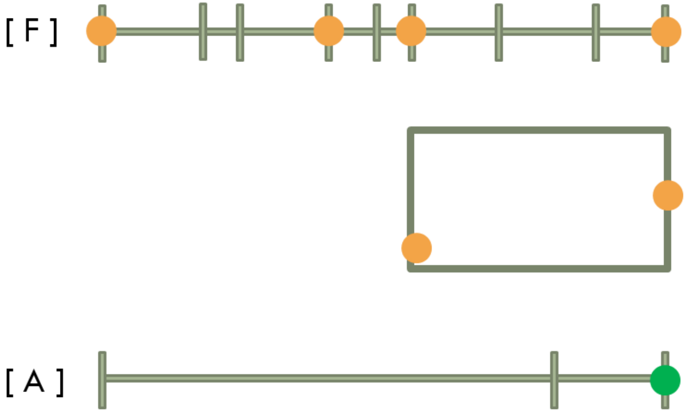
Adaptive Checkpointing

[F]

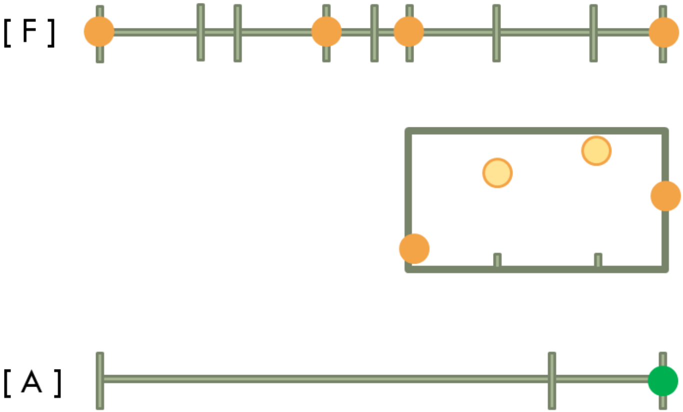


[A]

Adaptive Checkpointing

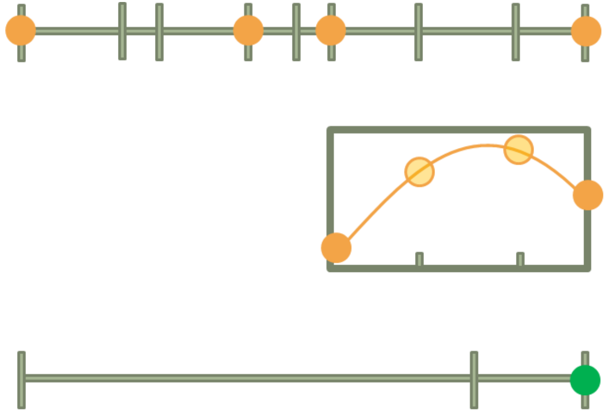


Adaptive Checkpointing



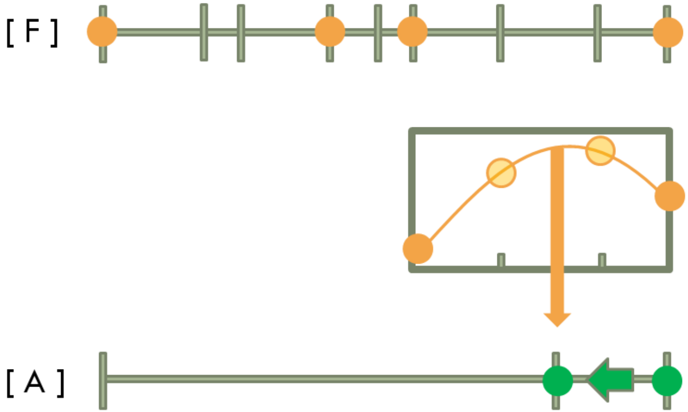
Adaptive Checkpointing

[F]



[A]

Adaptive Checkpointing



Adaptive Checkpointing