Inversion Using the `TSOpt` Framework

Marco Enriquez

The Rice Inversion Project (TRIP)
marco.enriquez@caam.rice.edu

January 30, 2009

Outline

- Simulation-driven optimization problems
- `TSOpt` ("Time Stepping for Optimization"):
  - How `TSOpt` works
  - `TSOpt`'s software architecture
  - `TSOpt`'s Services
- Using `RVL` and `Umin` in conjunction with `TSOpt`
- Example: Optimal Well Rate Allocation Problem

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)

TSOpt ("Time Stepping For Optimization")

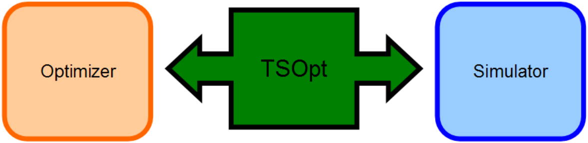
Motivating observation: for every simulation driven optimization problem, the solution process is (mostly) the same:

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

TSOpt is TRIP's "middle-ware" package. TSOpt:

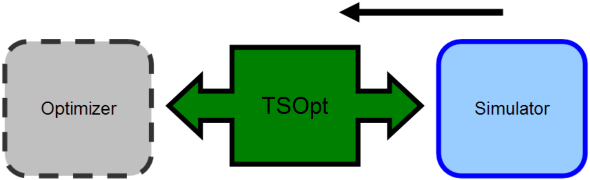
- abstracts commonalities among time-stepping methods
- provides a way for a simulation package to inter-operate with optimization algorithms
- supports use of the adjoint-state method

TSOpt ("Time Stepping For Optimization")



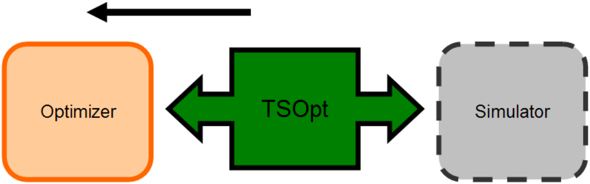
TSOpt ("Time Stepping For Optimization")

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

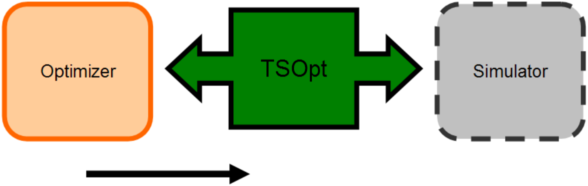


TSOpt ("Time Stepping For Optimization")

$$\nabla J(c), J(c), \ldots$$

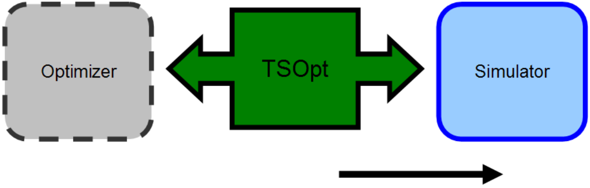


TSOpt ("Time Stepping For Optimization")



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

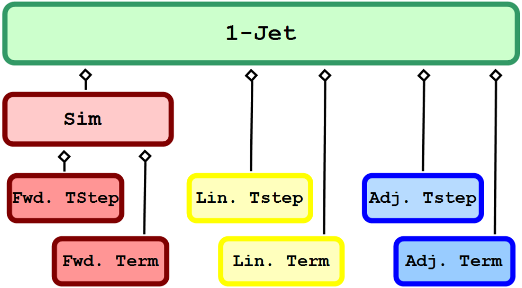
TSOpt ("Time Stepping For Optimization")



$$c = c^+$$

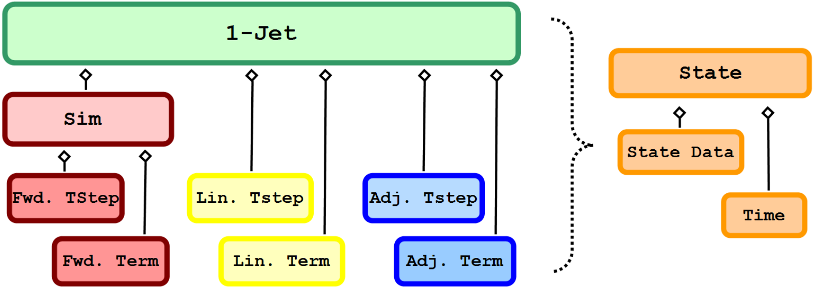
TSOpt's Components

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 (TRIP's `Umin` package)

TSOpt and the Adjoint-State (AS) Method

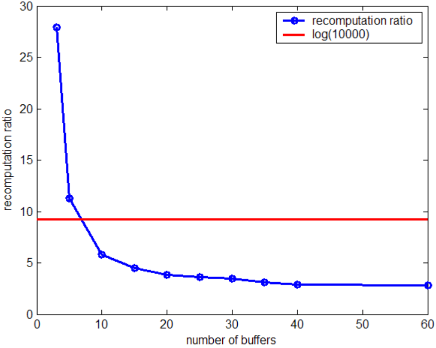
The AS method requires access to the reference simulation state history.

TSOpt implements the following strategies to address this:

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

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 $\langle Ax, y\rangle - \langle x, A^*y\rangle$

Using RVL and Umin

It is possible to use the TRIP software packages RVL and Umin to solve the simulation-driven optimization problem

RVL:

- collection of C++ classes expressing core concepts of calculus in Hilbert space (e.g. vector, operator, etc.)
- provides interfaces behind which to hide application-dependent implementation details

Umin:

- an extension of RVL
- TRIP's collection of unconstrained optimization algorithms

Using RVL and Umin

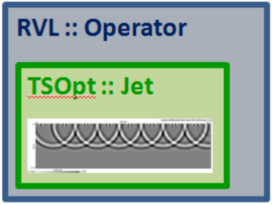
Idea: Use TSOpt jet object to create an RVL operator for use with optimization algorithms in Umin

Using RVL and Umin

Idea: Use TSOpt jet object to create an RVL operator for use with optimization algorithms in Umin

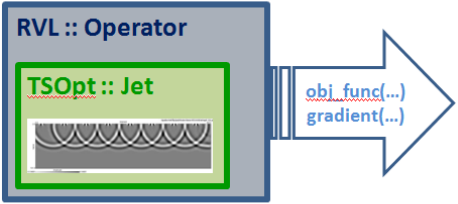
Using RVL and Umin

Idea: Use TSOpt jet object to create an RVL operator for use with optimization algorithms in Umin



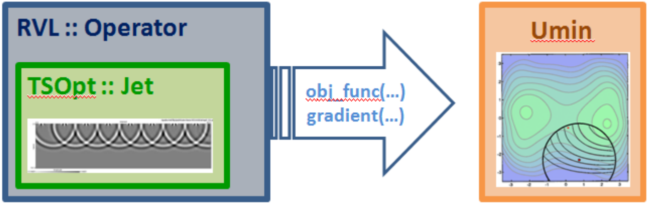
Using RVL and Umin

Idea: Use TSOpt jet object to create an RVL operator for use with optimization algorithms in Umin



Using RVL and Umin

Idea: Use TSOpt jet object to create an RVL operator for use with optimization algorithms in Umin



An Example: Optimal Well Rate Allocation

Consider the following simulation-driven optimization problem:

> Given the location of injecting and producing wells for a reservoir, **find the optimal well rate that maximizes revenue from oil production**, while penalizing water injection and production

To simulate interaction of oil and water in a reservoir, we use the **2D two-phase Black-Oil equations**, which we will treat as an *implicit constraint* ("black-box" formulation)

An Example: Optimal Well Rate Allocation

Mathematically, we state the optimal well rate allocation problem as:

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

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

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

Solving The State Equations

After using a Finite Volume and Backward Euler scheme, the fully discretized optimal control problem then takes the form of:

$$\min \qquad J_{\Delta t}(q) = \Delta t \sum_{k=1}^{N} l(t^k, s^k, q^k)$$

where s^{k+1} and p^{k+1} solve:

$$\begin{bmatrix} f(t^{k+1}, s_a^{k+1}, p^{k+1}, q^{k+1}) \\ g(t^{k+1}, s_a^{k+1}, p^{k+1}, q^{k+1}) \end{bmatrix} = \begin{bmatrix} q - Ap^{k+1} \\ D^{-1}(q_a - \tilde{A}p^{k+1}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_a^{k+1} - s_a^k}{\Delta t} \end{bmatrix}$$

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

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

The Adjoint Equations

For $k = N-1, \dots, 1$, simultaneously solve for the adjoint variables λ_s^k and λ_p^k in the following equation:

$$
\begin{aligned}
-\frac{\lambda_s^{k+1} - \lambda_s^k}{\Delta t} &= D_s f(\dots^k)^T \lambda_s^k - D_s g(\dots^k)^T \lambda_p^k - \nabla_s l(\dots^k) \\
0 &= -D_p f(\dots^k)^T \lambda_s^k + D_p g(\dots^k)^T \lambda_p^k
\end{aligned}
$$

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

$$
\nabla J(q)\delta q = \sum_{k=1}^{N} \Delta t [\nabla_q l(\cdot^k) - D_{q^k} f(\dots^k)^T \lambda_s^k + D_{q^k} g(\dots^k)^T \lambda_p^k]^T \delta q^k
$$

Implementation in TSOpt

Solving the optimal well rate allocation problem using TSOpt requires:

- **State type**: holds primary variables (p^k and s_a^k) and simulation time
- **Stack type**: holds a collection of simulation states
  - required for "save-all" or checkpointing strategies
  - must define typical stack operations `push_back()`, `pop()`, `size()`, etc.
- **Supporting Classes**: classes that support forward and adjoint simulators
  - Fluid: holds fluid properties, performs basic fluid-related computations
  - Wells: holds well location information and related calculations
  - Grid: holds grid and field information (size, porosity, permeability fields)

Implementation in TSOpt

- **Forward Simulator** (solves discretized Black-Oil equations)

$$\begin{bmatrix} q - Ap^{k+1} \\ D^{-1}(q_a - \tilde{A}p^{k+1}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_a^{k+1} - s_a^k}{\Delta t} \end{bmatrix}$$

Solution approach:

  - solve for p^{k+1} and s^{k+1} simultaneously using Newton-Raphson
  - linear system solves via UMFPACK

- **Adjoint Simulator**

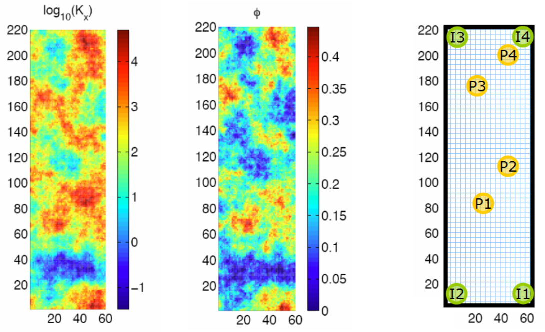
$$\begin{bmatrix} D_s f(\tilde{q}, p^*, s^*)^T & -D_s g(\tilde{q}, p^*, s^*)^T \\ -D_p f(\tilde{q}, p^*, s^*)^T & D_p g(\tilde{q}, p^*, s^*)^T \end{bmatrix} \begin{bmatrix} \lambda_s^{k+1} \\ \lambda_p^{k+1} \end{bmatrix} = \begin{bmatrix} \frac{\lambda_s^k - \lambda_s^{k+1}}{\Delta t} + \nabla_s l(\tilde{q}, p^*, s^*) \\ 0 \end{bmatrix}$$

Solution Approach:

  - solve for λ_p^{k+1} and λ_s^{k+1} (linear system solve via UMFPACK)

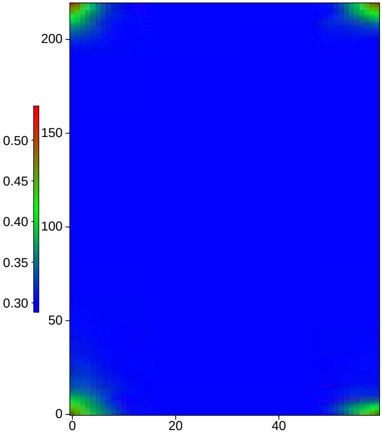
Simulation Information

- SPE10 data for porosity and permeability (left)
- Location of Injecting/Producing Wells (right)



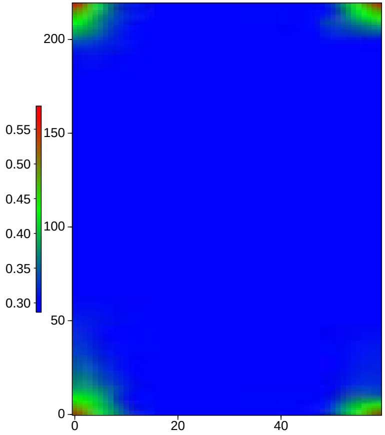
Reference Simulation Results

Saturation plot for $t = 25$ days

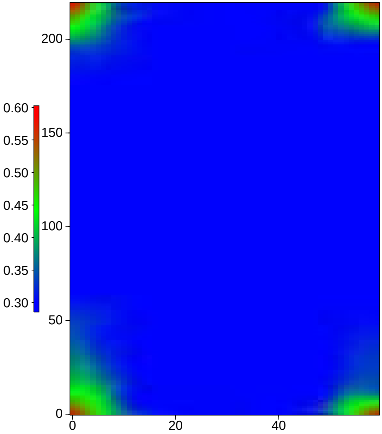


Reference Simulation Results

Saturation plot for $t = 50$ days

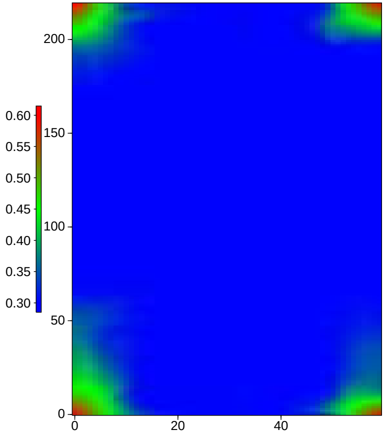


Saturation plot for $t = 75$ days



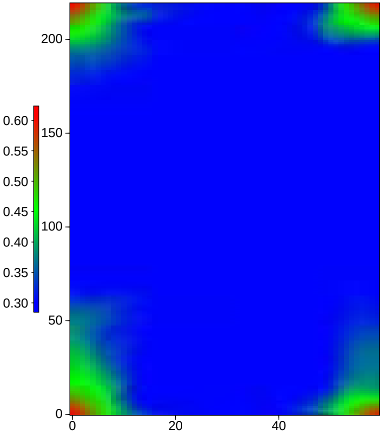
Reference Simulation Results

Saturation plot for $t = 100$ days



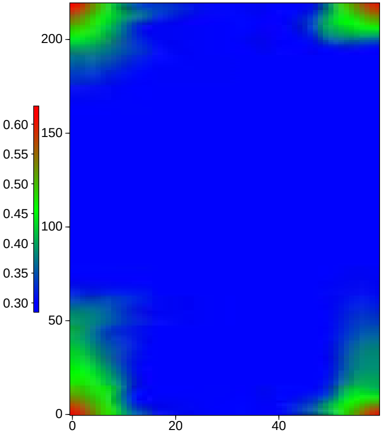
Reference Simulation Results

Saturation plot for $t = 125$ days



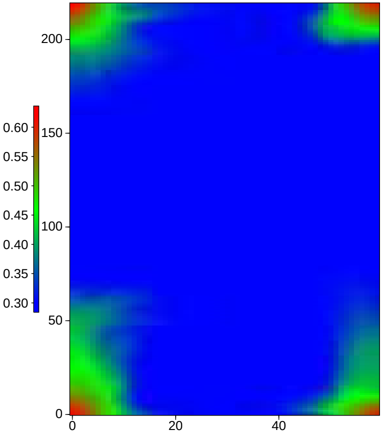
Reference Simulation Results

Saturation plot for $t = 150$ days



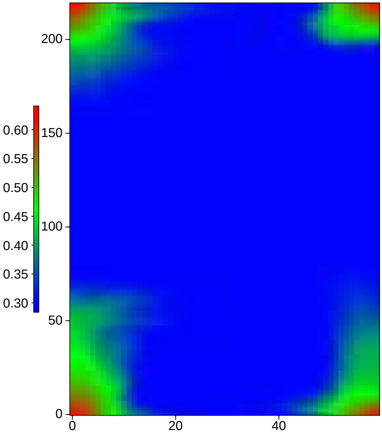
Reference Simulation Results

Saturation plot for $t = 175$ days

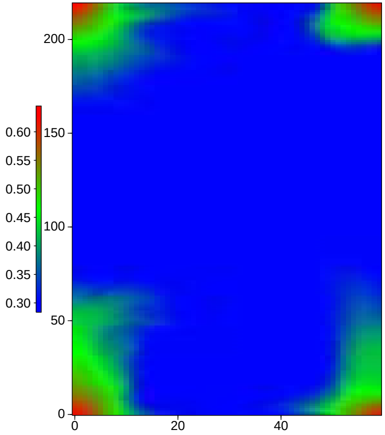


Reference Simulation Results

Saturation plot for $t = 200$ days

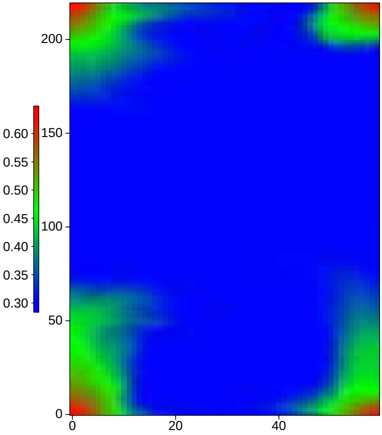


Saturation plot for $t = 225$ days



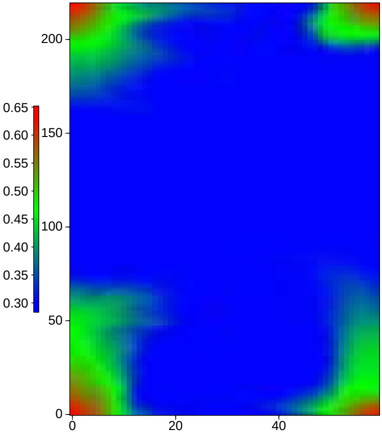
Reference Simulation Results

Saturation plot for $t = 250$ days



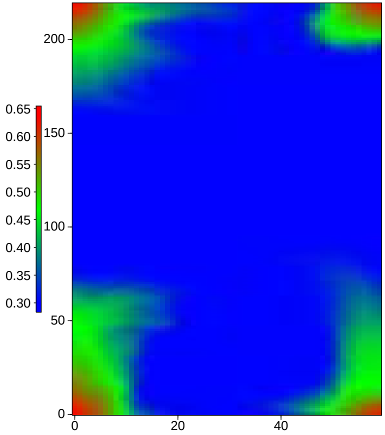
Reference Simulation Results

Saturation plot for $t = 275$ days

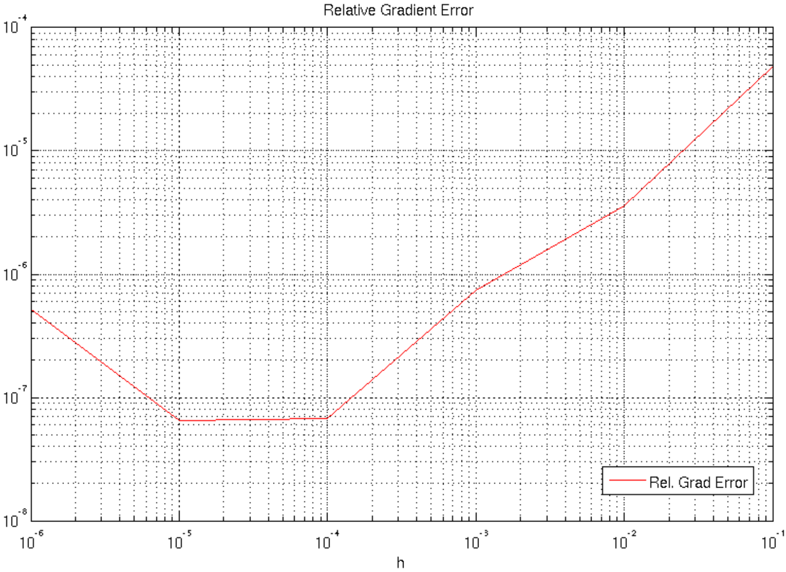


Reference Simulation Results

Saturation plot for $t = 300$ days



Gradient Quality



Inversion Results

How should the well rates be controlled for 300 days for the optimal well-rate allocation problem?

Simulation Information:

- LBFGS algorithm to perform optimization
- Simulation time range $= [0, 300]$, with $dt = 25$

Result:

- All wells increase their rate in an unbounded manner!

Need explicit constraints on the well rates for the problem to be well-posed

Reformulation of the Problem

Consider the following discretized problem, now with both (equality and inequality) explicit and implicit constraints:

$$\begin{aligned} \min \quad & J_{\Delta t}(q) = \Delta t \sum_{k=1}^{N} l(t^k, s^k, q^k) \\ s.t. \quad & e^T q^k = 0 \\ & q_{min} \leq q^k \leq q_{max} \, , \end{aligned}$$

where s^{k+1} and p^{k+1} solve:

$$\begin{bmatrix} q - A p^{k+1} \\ D^{-1}(q_a - \tilde{A} p^{k+1}) \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_a^{k+1} - s_a^k}{\Delta t} \end{bmatrix} .$$

Reformulation of Objective Function (WIP)

In order to address the new explicit constraints, we must either

- augment the objective function to include penalty terms (log-barrier, etc.)
- choose an SQP-type method to solve the problem

Conclusion

`TSOpt`:

- helps minimize code to solve a simulation-driven optimization problem
- minimizes code needed for checkpointing in adjoint calculation
- validates simulation relationships *before* inversion is attempted

`RVL` and `Umin` are available software packages for optimization

- minimal work is needed to link `TSOpt` with `RVL` and `Umin`

`TSOpt` used to solve the optimal well rate allocation problem

- augment objective function to incorporate explicit constraints
- simulator will be the base to study effects of adaptive time stepping for simulation-driven optimization problems

Acknowledgements

- K. Wiegand, A. El-Bakry, A. Singer (ExxonMobil URC)
- M. Heinkenschloss, W. Symes (Rice U.)
- NSF Rice VIGRE Fellowship