PDE-constrained Optimization Using PETSc/TAO

Presented to
ATPESC 2019 Participants

Alp Dener
Postdoctoral Appointee, MCS, ANL

Q Center, St. Charles, IL (USA)
Date 08/06/2019

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

What is optimization?

$$\underset{p}{\text{minimize}} \quad f(p)$$

- Optimization variables $p \in \mathbb{R}^n$
  - e.g.: boundary conditions, parameters, geometry
- Objective function $f\colon \mathbb{R}^n \to \mathbb{R}$
  - e.g.: lift, drag, pressure, temperature, stress, strain

What is optimization?

$$\underset{p}{\text{minimize}} \quad f(p)$$

- Simplification: $f(p)$ is minimized where $\nabla_p f(p) = 0$
- Gradient-free: Heuristic search through p space
- Gradient-based: Find search directions based on $\nabla_p f$

EXASCALE COMPUTING PROJECT

Why do we care?

We know a lot about how to solve the forward problem…

Inputs	Simulation	Outputs
• Boundary Conditions • Parameters • Geometry	• Fluid Dynamics • Continuum Mechanics • Heat Transfer	• Lift • Drag • Stress • Temperature

Why do we care?

We know a lot about how to solve the forward problem…

Inputs → Simulation → Outputs

Inputs
- Boundary Conditions
- Parameters
- Geometry

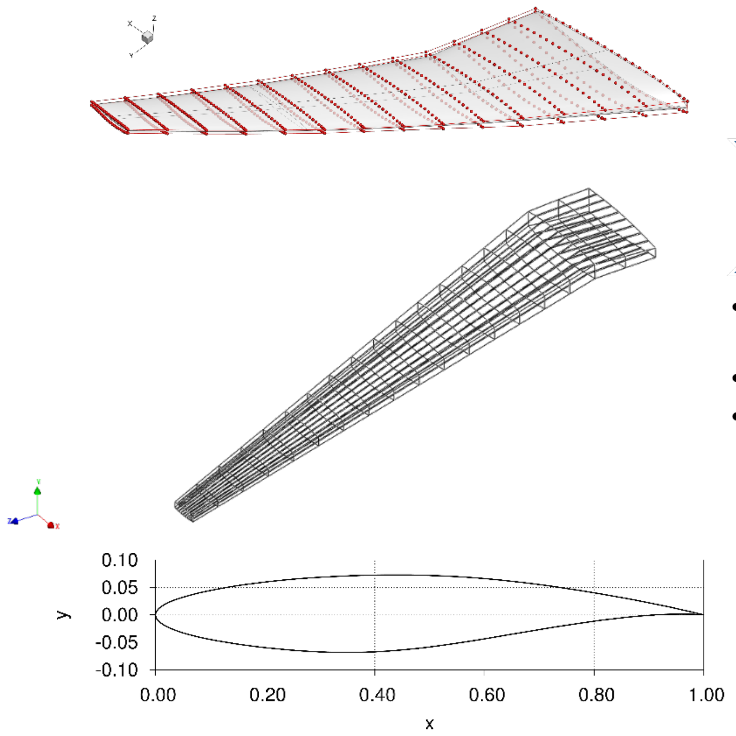
Simulation
- Fluid Dynamics
- Continuum Mechanics
- Heat Transfer

Outputs
- Lift
- Drag
- Stress
- Temperature

Control Decision Design ← Optimization ← Objective

…many scientific questions present as inverse problems!

Why do we care?

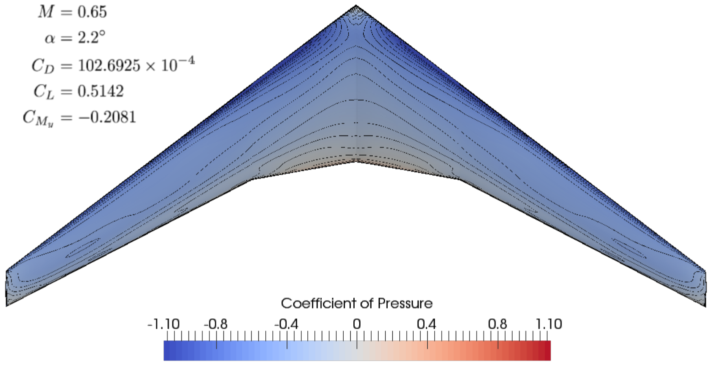


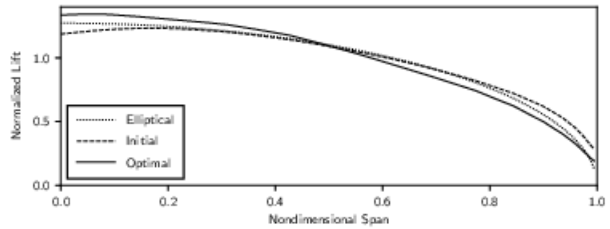
We know a lot about how to solve the forward problem…

Inputs	Simulation	Outputs
• Boundary Conditions • Parameters • Geometry	• Fluid Dynamics • Continuum Mechanics • Heat Transfer	• Lift • Drag • Stress • Temperature

Control Decision Design ← Optimization ← Objective

…many scientific questions present as inverse problems!





EXASCALE COMPUTING PROJECT

Outline

- Intro to PDE-constrained Optimization
  - Full-space formulation
  - Reduced-space formulation
- Introduction to TAO
  - Sample main program
  - User/problem function callback
- Sensitivity Analysis
  - Finite difference method
  - Adjoint method
- Hands-on Example: Boundary Control w/ 2D Laplace Equation

PDE-constrained Optimization

$$\begin{aligned} \underset{p,u}{\text{minimize}} \quad & f(p,u) \\ \text{subject to} \quad & R(p,u) = 0 \end{aligned}$$

- Optimization variables $p \in \mathbb{R}^n$
- State variables $u \in \mathbb{R}^m$
- State equations R: $\mathbb{R}^{n+m} \rightarrow \mathbb{R}^m$
- “Full-space” formulation – optimization includes both optimization and state variables

First-order Optimality Conditions

- Construct the Lagrangian where $\lambda \in \mathbb{R}^m$

$$\mathcal{L}(p, u, \lambda) = f(p, u) + \lambda^T R(p, u)$$

- Differentiate w.r.t. every input for first-order optimality

$$\begin{aligned}
\nabla_p \mathcal{L} &= \frac{\partial f}{\partial p} + \lambda^T \frac{\partial R}{\partial p} &= 0 \\
\nabla_u \mathcal{L} &= \frac{\partial f}{\partial u} + \lambda^T \frac{\partial R}{\partial u} &= 0 \\
\nabla_\lambda \mathcal{L} &= R(u, p) &= 0
\end{aligned}$$

- Also known as the Karush-Kuhn-Tucker (KKT) conditions

Solving the Problem

- Apply Newton's method to the KKT conditions

For k = 0, 1, 2, …

Convergence check (i.e., $\|\nabla_p \mathcal{L}\| \leq \varepsilon_p$ and $\|R\| \leq \varepsilon_u$)

Solve $$\begin{bmatrix} \nabla^2_{pp}\mathcal{L} & \nabla^2_{up}\mathcal{L} & \frac{\partial R}{\partial p}^T \\ \nabla^2_{pu}\mathcal{L} & \nabla^2_{uu}\mathcal{L} & \frac{\partial R}{\partial u}^T \\ \frac{\partial R}{\partial p} & \frac{\partial R}{\partial u} & 0 \end{bmatrix}_k \begin{pmatrix} \Delta p \\ \Delta u \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla_p \mathcal{L} \\ -\nabla_u \mathcal{L} \\ -R(p,u) \end{pmatrix}_k$$

Step acceptance with globalization (e.g., line search)

EXASCALE COMPUTING PROJECT

Solving the Problem

- Apply Newton's method to the KKT conditions

For k = 0, 1, 2, …

Convergence check (i.e., $\|\nabla_p \mathcal{L}\| \leq \varepsilon_p$ and $\|R\| \leq \varepsilon_u$)

Solve $$\begin{bmatrix} \nabla^2_{pp}\mathcal{L} & \nabla^2_{up}\mathcal{L} & \frac{\partial R}{\partial p}^T \\ \nabla^2_{pu}\mathcal{L} & \nabla^2_{uu}\mathcal{L} & \frac{\partial R}{\partial u}^T \\ \frac{\partial R}{\partial p} & \frac{\partial R}{\partial u} & 0 \end{bmatrix}_k \begin{pmatrix} \Delta p \\ \Delta u \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla_p \mathcal{L} \\ -\nabla_u \mathcal{L} \\ -R(p, u) \end{pmatrix}_k$$

Conjugate Gradient
Quasi-Newton
Newton-Krylov
…

Step acceptance with globalization (e.g., line search)

Solving the Problem

$$\begin{bmatrix} \nabla^2_{pp}\mathcal{L} & \nabla^2_{up}\mathcal{L} & \frac{\partial R}{\partial p}^T \\ \nabla^2_{pu}\mathcal{L} & \nabla^2_{uu}\mathcal{L} & \frac{\partial R}{\partial u}^T \\ \frac{\partial R}{\partial p} & \frac{\partial R}{\partial u} & 0 \end{bmatrix}_k \begin{pmatrix} \Delta p \\ \Delta u \\ \Delta \lambda \end{pmatrix} = \begin{pmatrix} -\nabla_p \mathcal{L} \\ -\nabla_u \mathcal{L} \\ -R(p,u) \end{pmatrix}_k$$

- **Full-space** formulation
  - PDE solution tightly coupled with optimization
  - Avoid the cost of a complete PDE solution at every optimization iteration
  - Large saddle-point problem – difficult to solve

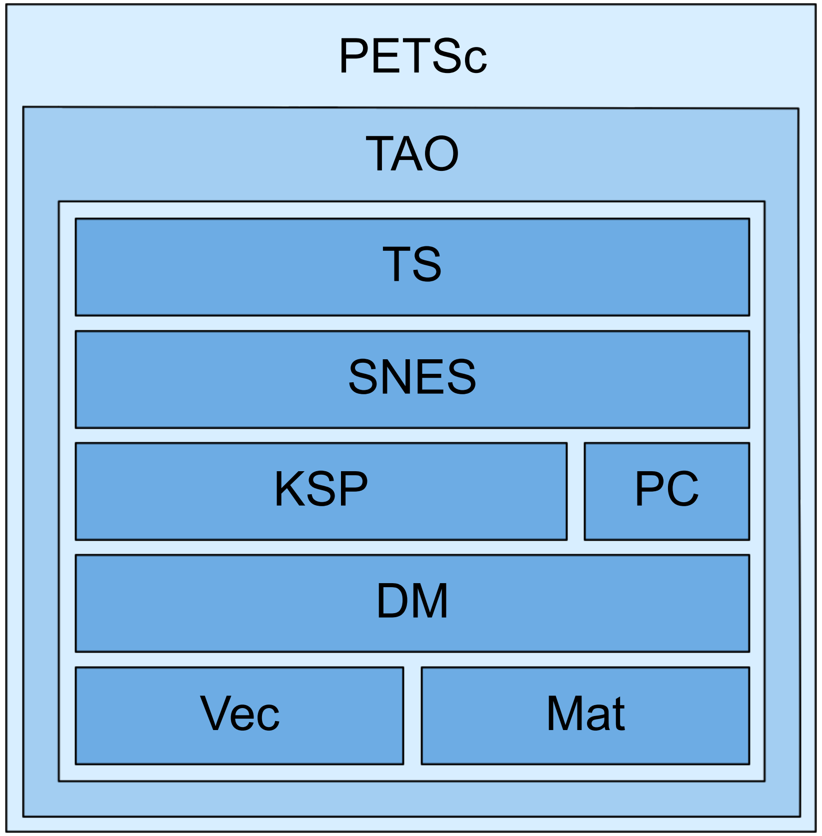
An Alternative Approach

$$\underset{p}{\text{minimize}} \quad f(p, u(p))$$

- **Reduced-space** formulation
  - PDE-constraint is eliminated via implicit function theorem
  - Optimization algorithm and PDE solver are independent
  - Each objective evaluation requires a full PDE solution

Toolkit for Advanced Optimization (TAO)

- General-purpose continuous optimization toolbox for large-scale problems
  - Parallel (via PETSc data structures)
  - Gradient-based
  - Bound-constrained
  - Nonlinear constraint support under development
- **Support for reduced-space PDE-constrained optimization**
- Distributed with PETSc (https://www.mcs.anl.gov/petsc/)
- Similar packages:
  - Rapid Optimization Library (https://trilinos.github.io/rol.html)
  - HiOP (https://github.com/LLNL/hiop)



TAO: The Basics

- Sample main program

```
AppCtx user;
Tao tao;
Vec P;

PetscInitialize( &argc, &argv,(char *)0,help );
VecCreateMPI(PETSC_COMM_WORLD, user.n, user.N, &P);
VecSet(P, 0.0);

TaoCreate(PETSC_COMM_WORLD, &tao);
TaoSetType(tao, TAOBQNLS); /* BQNLS: quasi-Newton line search */
TaoSetInitialVector(tao, P);
TaoSetObjectiveAndGradientRoutine(tao, FormFunctionGradient, (void*) &user);
TaoSetFromOptions(tao);
TaoSolve(tao);

VecDestroy(&P);
TaoDestroy(&tao);
PetscFinalize();
```

TAO: The Basics

- User provides function for problem implementation

```
AppCtx user;
Tao tao;
Vec P;

PetscInitialize( &argc, &argv,(char *)0,help );
VecCreateMPI(PETSC_COMM_WORLD, user.n, user.N, &P);
VecSet(P, 0.0);

TaoCreate(PETSC_COMM_WORLD, &tao);
TaoSetType(tao, TAOBQNLS); /* BQNLS: quasi-Newton line search */
TaoSetInitialVector(tao, P);
TaoSetObjectiveAndGradientRoutine(tao, FormFunctionGradient, (void*) &user);
TaoSetFromOptions(tao);
TaoSolve(tao);

VecDestroy(&P);
TaoDestroy(&tao);
PetscFinalize();
```

TAO: User Function

- User function computes objective and gradient

```
typedef struct {
   /* user-created context for storing application data */
} AppCtx;

PetscErrorCode FormFunctionGradient(Tao tao, Vec P, PetscReal *fcn, Vec G, void *ptr) {
   AppCtx *user = (AppCtx*)ptr;
   const PetscScalar *pp;
   PetscScalar *gg;

   VecGetArrayRead(P, &pp);
   VecGetArray(G, &gg);
   /* USER TASK: Compute objective function and store in fcn */
   /* USER TASK: Compute compute gradient and store in gg */
   VecRestoreArrayRead(P, &pp);
   VecRestoreArray(G, &gg);

   return 0;
}
```


EXASCALE COMPUTING PROJECT

TAO: User Function

- **State/PDE solution:**

  Solve $R(p, u) = 0$ for $u(p)$ at new p

- **Objective evaluation:**

  Compute $f(p, u(p))$

- **Sensitivity analysis:**

  Compute $\mathrm{G} = \nabla_p f$ at p and $u(p)$

TAO: User Function

- **State/PDE solution:**
  Solve $R(p,u) = 0$ for $u(p)$ at new p

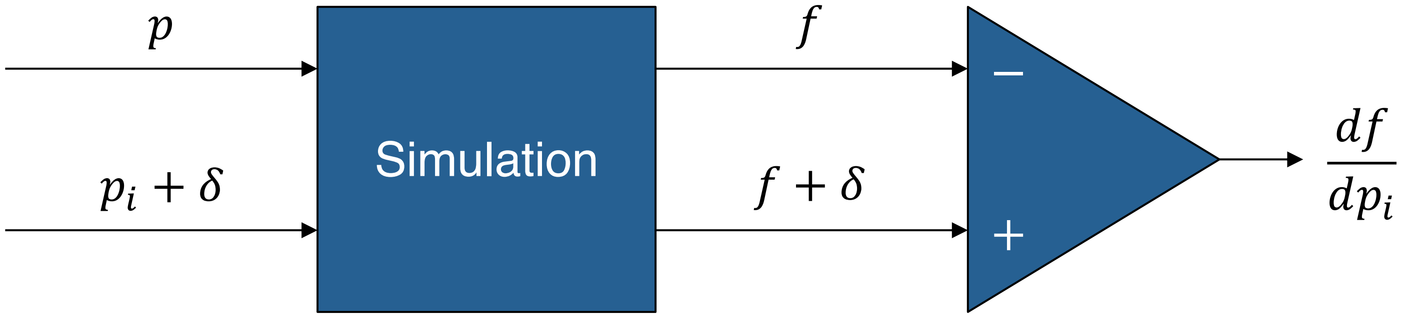
- **Objective evaluation:**
  Compute $f(p, u(p))$

- **Sensitivity analysis:**
  Compute $G = \nabla_p f$ at p and $u(p)$

- Necessary for gradient-based optimization
- Converges to the local optimum faster than gradient-free methods (i.e., fewer PDE solutions)
- Types:
  - Finite difference
  - Discrete adjoint method

EXASCALE COMPUTING PROJECT

Sensitivity Analysis: Finite Difference



- Easy to implement
  - Only requires function evaluations
- Inefficient for large numbers of optimization variables
- Step-size dilemma – truncation error vs. subtractive cancellation

EXASCALE COMPUTING PROJECT

Sensitivity Analysis: Discrete Adjoint Method

- Recall the first-order optimality conditions for PDE-constrained optimization

$$
\begin{aligned}
\nabla_p \mathcal{L} &= \frac{\partial f}{\partial p} + \lambda^T \frac{\partial R}{\partial p} &= 0 \quad (1)\\
\nabla_u \mathcal{L} &= \frac{\partial f}{\partial u} + \lambda^T \frac{\partial R}{\partial u} &= 0 \quad (2)\\
\nabla_\lambda \mathcal{L} &= R(u, p) &= 0 \quad (3)
\end{aligned}
$$

- Reduced-space formulation solves equations (2) and (3) fully at each optimization iteration and substitutes into equation (1)

Sensitivity Analysis: Discrete Adjoint Method

1. **State/PDE Solution:**

   Solve $R(p,u) = 0$ for u at new p

2. **Adjoint Solution:**

   Solve $\left(\frac{\partial R}{\partial u}\right)^T \lambda = -\frac{\partial f}{\partial u}$ for λ at new p and $\mathrm{u}(p)$

3. **Gradient Assembly:**

   Compute $\mathrm{G} = \nabla_p f = \frac{\partial f}{\partial p} + \lambda^T \frac{\partial R}{\partial p}$

Sensitivity Analysis: Discrete Adjoint Method

1. **State/PDE Solution:**

   Solve $R(p,u) = 0$ for u at new p ⟵ Solve $\nabla_\lambda \mathcal{L} = 0$ (eqn. 3)

2. **Adjoint Solution:**

   Solve $\left(\frac{\partial R}{\partial u}\right)^T \lambda = -\frac{\partial f}{\partial u}$ for λ at new p and $\mathrm{u}(p)$ ⟵ Solve $\nabla_u \mathcal{L} = 0$ (eqn. 2)

3. **Gradient Assembly:**

   Compute $\mathrm{G} = \nabla_p f = \frac{\partial f}{\partial p} + \lambda^T \frac{\partial R}{\partial p}$ ⟵ Evaluate $\nabla_p \mathcal{L}$ (eqn. 1)

Sensitivity Analysis: Discrete Adjoint Method

- Computational cost is independent of the number of optimization variables
  - One linear system solution for each scalar function

- Requires transpose operators for the PDE Jacobians (w.r.t. both state and optimization variables)

- PETSc/TS (Time Steppers) package implements checkpointing for backwards-in-time adjoint solutions
  - Not covered in this tutorial

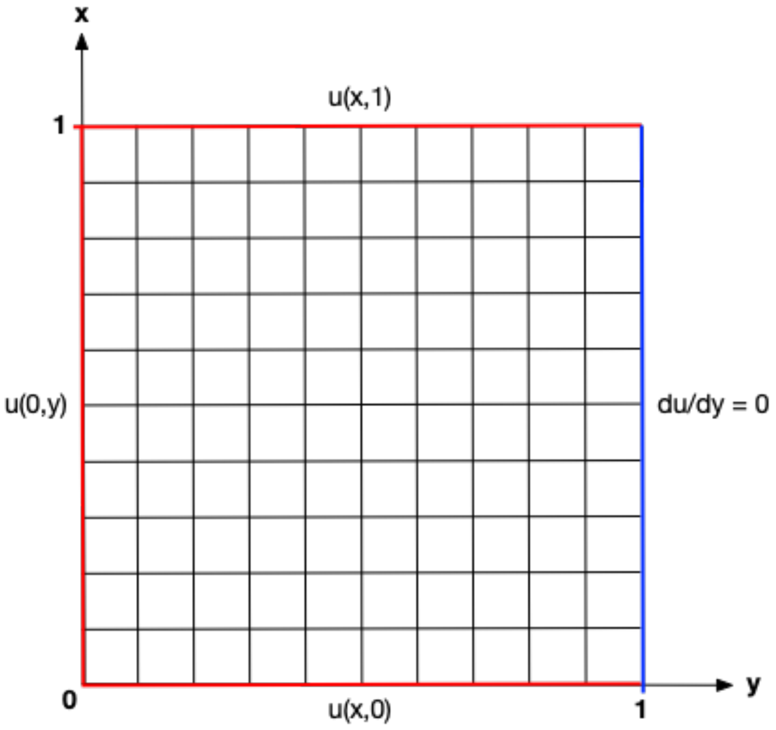
Hands-on Example: Boundary Control w/ 2D Laplace Eqn

$$\underset{p}{\text{minimize}} \quad \frac{1}{2}\int_0^1 (u(1,y) - u_{target})^2 dy$$

$$\text{governed by} \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad \forall\, x, y \in (0,1)$$

$$\left.\frac{\partial u}{\partial x}\right|_{u(1,y)} = 0, \quad \forall\, y \in (0,1)$$

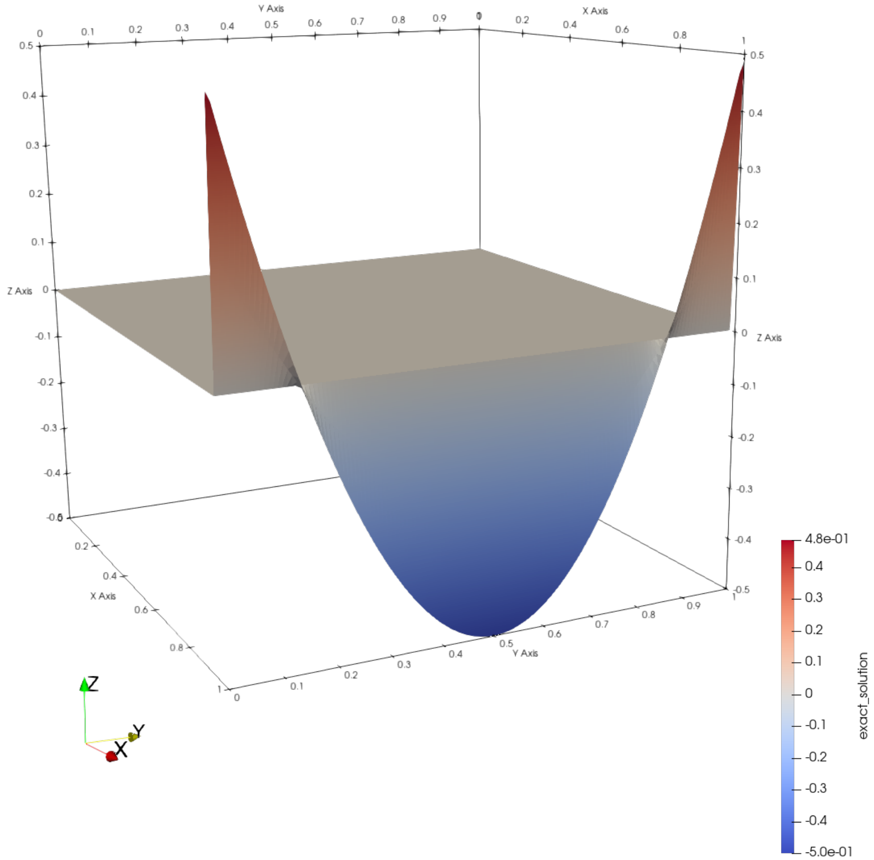
$$p = \begin{bmatrix} u(x,0) & u(x,1) & u(0,y) \end{bmatrix}^T$$



- Control left, top and bottom Dirichlet bounds to recover target solution at the right boundary

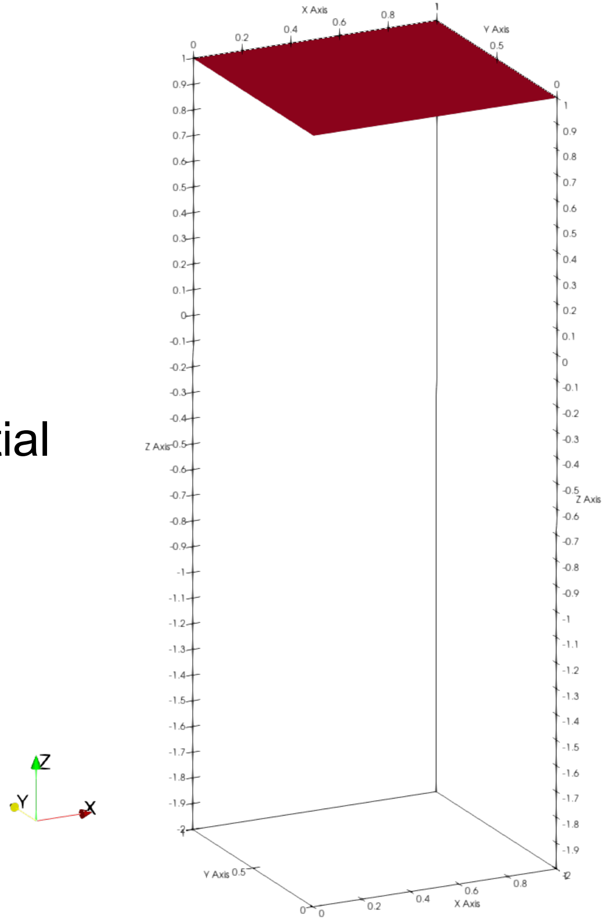
Hands-on Example: Boundary Control w/ 2D Laplace Eqn

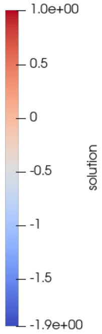
- PDE solution, objective function and gradient evaluations implemented with AMReX (https://amrex-codes.github.io/amrex/)
- Target solution set to $u_{target} = 4(y - 0.5)^2 - 0.5$
- Lapace equation is self-adjoint
  - PDE Jacobian is symmetric
- Command line options:
  - Problem size: `-nx 128`
  - Switch on finite difference gradient: `-fd`

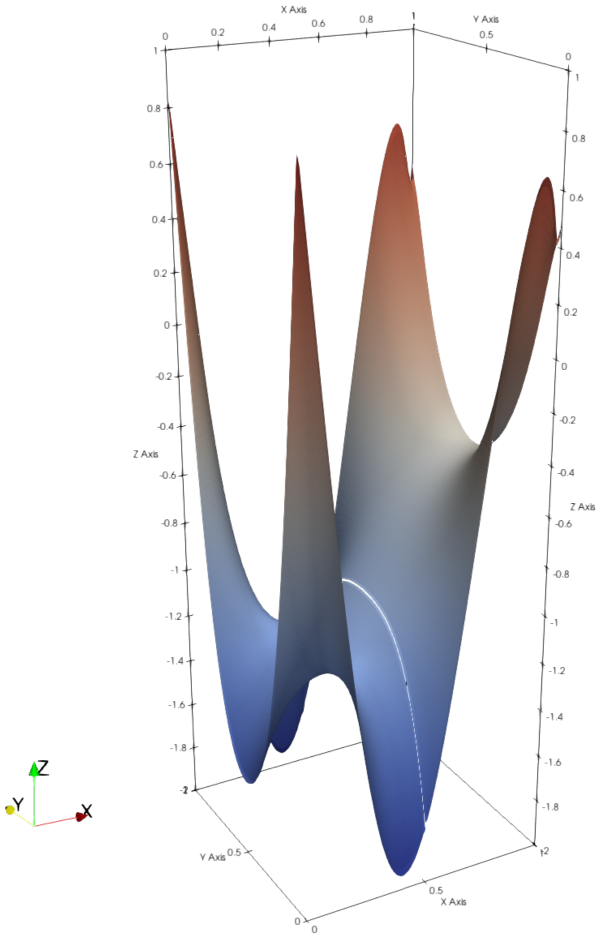


Hands-on Example: Boundary Control w/ 2D Laplace Eqn

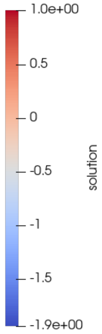
Initial







Final



Hands-on Example: Boundary Control w/ 2D Laplace Eqn

```
$ ./main2d.gnu.MPI.ex inputs -tao_monitor -tao_ls_type armijo -tao_fmin 1e-6 -tao_gatol 1e-12
```

```
AMReX (19.08) initialized
  0 TAO,  Function value: 0.00564792,  Residual: 4.57574e-08
  1 TAO,  Function value: 0.00564792,  Residual: 4.57574e-08
  2 TAO,  Function value: 0.000351831,  Residual: 6.72001e-09
  3 TAO,  Function value: 0.000104118,  Residual: 4.5399e-09
  4 TAO,  Function value: 4.39909e-06,  Residual: 5.62804e-10
  5 TAO,  Function value: 3.20011e-06,  Residual: 3.18434e-10
  6 TAO,  Function value: 2.90562e-06,  Residual: 3.67294e-10
  7 TAO,  Function value: 2.82687e-06,  Residual: 3.54986e-10
  8 TAO,  Function value: 2.74912e-06,  Residual: 3.45265e-10
  9 TAO,  Function value: 2.74912e-06,  Residual: 3.45265e-10
 10 TAO,  Function value: 2.5271e-06,  Residual: 1.99102e-10
 11 TAO,  Function value: 2.41122e-06,  Residual: 1.43435e-10
 12 TAO,  Function value: 2.10344e-06,  Residual: 8.34685e-11
 13 TAO,  Function value: 1.78759e-06,  Residual: 1.11393e-10
 14 TAO,  Function value: 1.30814e-06,  Residual: 8.4585e-11
 15 TAO,  Function value: 1.10837e-06,  Residual: 9.14759e-11
 16 TAO,  Function value: 1.08865e-06,  Residual: 9.52258e-11
 17 TAO,  Function value: 8.87623e-07,  Residual: 1.29118e-10
TaoSolve() duration: 1316289 microseconds
[The  Pinned Arena] space (MB) used spread across MPI: [8 ... 8]
AMReX (19.08) finalized
```

Hands-on Example: Boundary Control w/ 2D Laplace Eqn

- **Tutorial goals:**
  - Compare computational cost of finite difference gradient to the adjoint method
  - Verify that the cost of the adjoint method is (mostly) independent of the number of optimization variables
  - Solve the problem with different TAO algorithms
  - Interpret the TAO monitor output and assess convergence

Take Away Messages

- **PDE-constrained optimization does not need to be intimidating!**
  - PETSc/TAO provides interfaces and algorithms that work with reduced-space formulations
  - PETSc/TAO can compute gradients and Hessians automatically with finite differencing
  - PETSc data structures are easy to couple with most PDE solvers (e.g., AMReX)
- **The adjoint method is ideal for sensitivity analysis**
  - Some PDE solvers already have necessary building blocks
  - Possible to take implementation shortcuts in self-adjoint problems
- **PETSc/TAO offers parallel optimization algorithms for large-scale problems**
  - Optimization data structures are duplicated from user-generated PETSc vectors
  - User has full control over parallel distribution and vector type

Acknowledgements

AMReX: https://amrex-codes.github.io/amrex/

PETSc/TAO: https://www.mcs.anl.gov/petsc/

Special thanks to Don Willcox for support in developing the hands-on example code.

Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program and the Exascale Computing Project funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research.