Numerical Optimization Using PETSc/TAO

Presented to
ATPESC 2020 Participants

Alp Dener
Postdoctoral Appointee, MCS, ANL

08/04/2020

U.S. DEPARTMENT OF ENERGY | Office of Science

ATPESC Numerical Software Track

EXASCALE COMPUTING PROJECT

Argonne NATIONAL LABORATORY · BERKELEY LAB · Sandia National Laboratories · Rensselaer · SMU

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, max stress, total energy, error norms, etc.

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
  - Easy to use, no sensitivity analysis required

- **Gradient-based:** Find search directions based on $\nabla_p f$
  - Converges to local minima with significantly fewer function evaluations than gradient-free methods

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

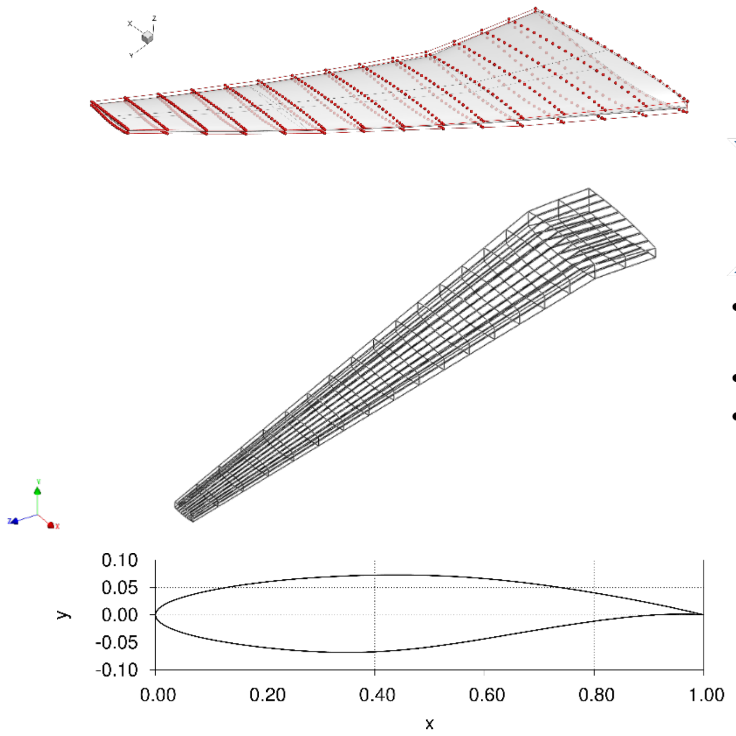
Simulation:
- Fluid Dynamics
- Continuum Mechanics
- Heat Transfer

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**

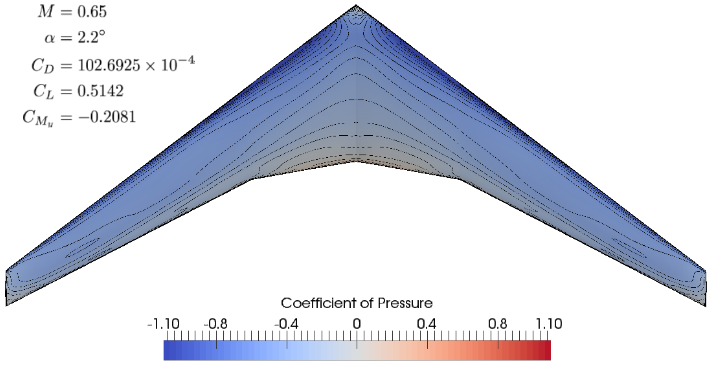
- Boundary Conditions
- Parameters
- Geometry

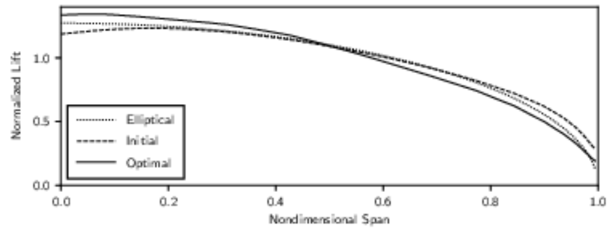
- Fluid Dynamics
- Continuum Mechanics
- Heat Transfer

- Lift
- Drag
- Stress
- Temperature

Control Decision Design ← **Optimization** ← **Objective**

…many scientific questions present as inverse problems!





EXASCALE COMPUTING PROJECT

Outline

- Introduction to Gradient-Based Optimization
  - Sequential Quadratic Programming
  - Sensitivity Analysis
- Introduction to TAO
  - Sample main program
  - User/problem callback function
- Hands-on Examples: Multidimensional Rosenbrock

Intro to Numerical Optimization

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

- Optimization variables $p \in \mathbb{R}^n$
- Objective function $f: \mathbb{R}^n \rightarrow \mathbb{R}$
- Local minima where gradient is zero (optimality condition)
- Optimality condition is **necessary** but not sufficient
  - Other stationary points (e.g., maxima) also satisfy $\nabla_p f(p) = 0$

Sequential Quadratic Programming

for k=0,1,2,... **do**

$$\min_d \quad f_k + d^T g_k + 0.5 d^T H_k d$$

$$\min_\alpha \quad \Phi(\alpha) = f(p_k + \alpha d)$$

$$p_{k+1} \leftarrow p_k + \alpha d$$

end for

- Solution at k^{th} iteration p_k
- Gradient $g_k = \nabla_p f(p_k)$
- Hessian $H_k = \nabla^2_{pp} f(p_k)$
- Search direction $d \in \mathbb{R}^n$
- Step length α

- Replace original problem with a sequence of quadratic subproblems
  - Solution given by $d = -H_k^{-1} g\_k$
- Line search maintains consistency between local quadratic model and global nonlinear function (globalization)
  - Avoids undesirable stationary points

Sequential Quadratic Programming

$$
\begin{aligned}
&\textbf{for } k=0,1,2,\ldots \textbf{ do}\\
&\quad \min_d \quad f_k + d^T g_k + 0.5 d^T H_k d\\
&\quad \min_\alpha \quad \Phi(\alpha) = f(p_k + \alpha d)\\
&\quad p_{k+1} \leftarrow p_k + \alpha d\\
&\textbf{end for}
\end{aligned}
$$

- Solution at k^{th} iteration p_k
- Gradient $g_k = \nabla_p f(p_k)$
- Hessian $H_k = \nabla^2_{pp} f(p_k)$
- Search direction $d \in \mathbb{R}^n$
- Step length α

- Different approximations to search direction yield different algorithms
  - **Newton's method:** $d = -H_k^{-1} g_k$, no approximation
  - **Quasi-Newton:** $d = -B_k g_k$ with $B_k \approx H_k^{-1}$ based on a Secant approximation
  - **Conjugate Gradient:** $d_k = -g_k + \beta_k d_{k-1}$ with β defining different CG updates
  - **Gradient Descent:** $d = -g_k$, replace Hessian with identity

PDE-Constrained Optimization

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

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

Full-Space Formulation

- State variables $u \in \mathbb{R}^m$
- State equations $R: \mathbb{R}^{n+m} \rightarrow \mathbb{R}^m$

Reduced-Space Formulation

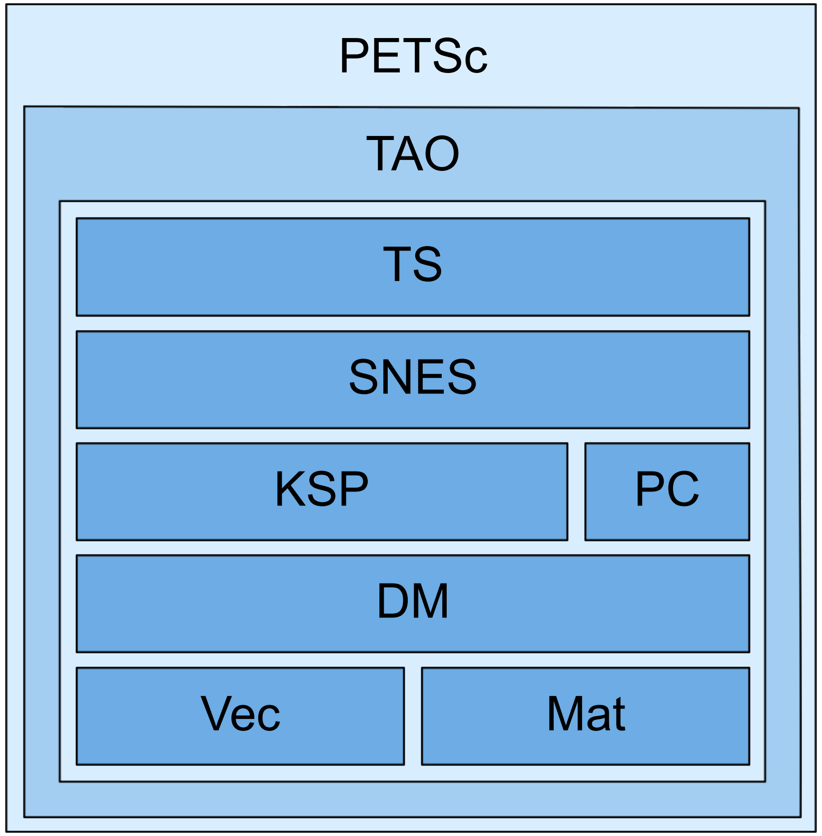
- State variables are implicit functions of parameters

- Reduced-space formulation enables use of conventional unconstrained optimization algorithms to solve PDE-constrained problems
- Each reduced-space function evaluation requires a full PDE solution
- See ATPESC 2019 lesson for more details

PETSc TAO

Toolkit for Advanced Optimization

- General-purpose continuous optimization toolbox for large-scale problems
  - Parallel (via PETSc data structures)
  - Gradient-based
  - Bound-constrained
  - Nonlinear/general constraint support under development
  - PDE-constrained problems w/ reduced-space formulation
- 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);
TaoSetObjectiveRoutine(tao, FormFunction, &user);
TaoSetGradientRoutine(tao, FormGradient, &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);
TaoSetObjectiveRoutine(tao, FormFunction, &user);
TaoSetGradientRoutine(tao, FormGradient, &user);
TaoSetFromOptions(tao);
TaoSolve(tao);

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

TAO: User Function

- User functions compute objective and gradient

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

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

    VecGetArrayRead(P, &pp);

    /* USER TASK: Compute objective function and store in fcn */

    VecRestoreArrayRead(P, &pp);

    return 0;
}
```

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

    VecGetArrayRead(P, &pp);
    VecGetArray(G, &gg);

    /* USER TASK: Compute compute gradient and store in gg */

    VecRestoreArrayRead(P, &pp);
    VecRestoreArray(G, &gg);

    return 0;
}
```

TAO: User Function

- **Objective evaluation:**
  - Compute $f(p)$ at given p

- **Sensitivity analysis:**
  - Compute $\mathrm{G} = \nabla_p f$ at given p

- **(ADVANCED) Second-order Methods:**
  - Compute $H = \nabla_p^2 f$ at given p
  - Use `TaoSetHessian()` interface

TAO: User Function

- **Objective evaluation:**
  - Compute $f(p)$ at given p

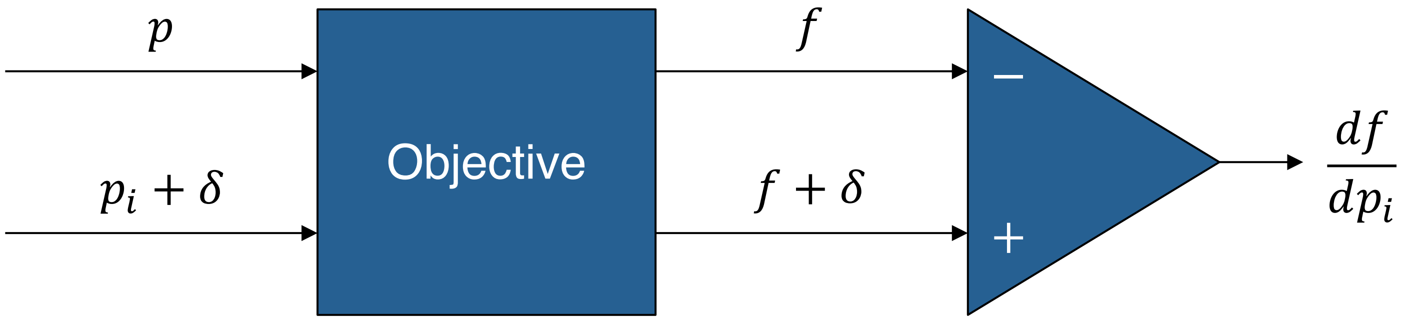
- **Sensitivity analysis:**
  - Compute $G = \nabla_p f$ at given p

→

- Necessary for gradient-based optimization algorithms
- Types:
  - Numerical
  - Analytical

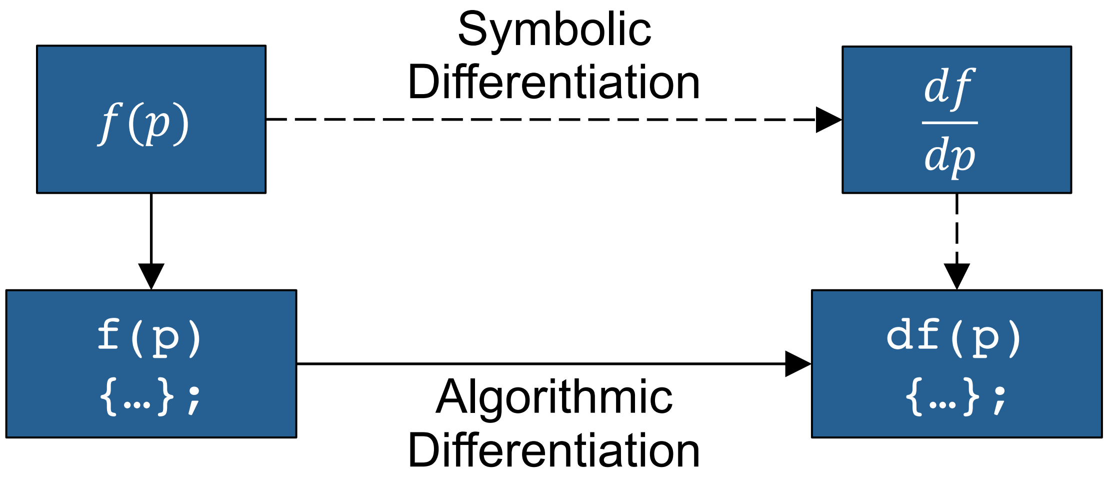
- **(ADVANCED) Second-order Methods:**
  - Compute $H = \nabla_p^2 f$ at given p
  - Use `TaoSetHessian()` interface

Sensitivity Analysis: Numerical Differentiation



- Finite-difference method is easy to implement
  - Only requires function evaluations
- Inefficient for large numbers of optimization variables
- Step-size dilemma – truncation error vs. subtractive cancellation
- TAO provides automatic FD gradient and Hessian evaluations

Sensitivity Analysis: Analytical Differentiation

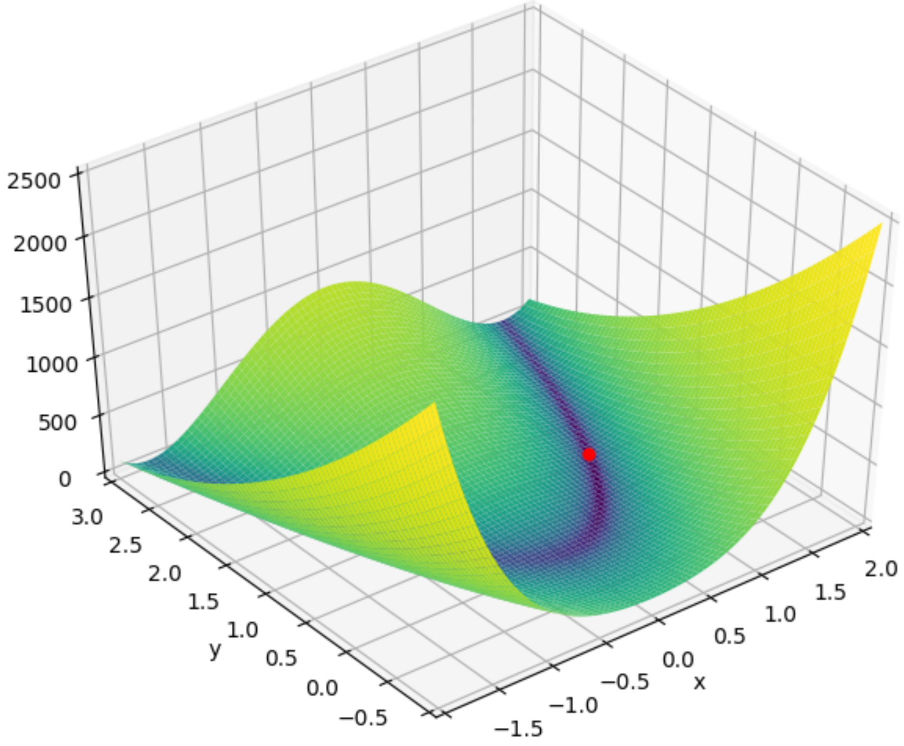


- **Symbolic** – manual hand-derived gradient, typically not applicable to simulation-based / HPC applications
- **Algorithmic** – source code transformation or operator overloading via AD tool/library (i.e., chain rule!)

- Computational cost is (mostly) independent of the number of optimization variables
- Implementation difficulty increases with function complexity
  – PDE-constrained problems need to implement the **adjoint method**
  – See ATPESC 2019 lesson for more details

Hands-on Example: Multidimensional Rosenbrock

$$\underset{p}{\text{minimize}} \quad f(p) = (1 - p_1)^2 + 100(p_2 - p_1^2)^2$$



- Original 2D version has a global minimum at $p = (1, 1)$
- Also called the "banana function"
- Canonical test problem for optimization algorithms
- Easy to find the valley, difficult to traverse it towards the solution

EXASCALE COMPUTING PROJECT

Hands-on Example: Multidimensional Rosenbrock

$$\underset{p}{\text{minimize}} \quad f(p) = \sum_{i=1}^{N-1} (1 - p_i)^2 + 100(p_{i+1} - p_i^2)^2$$

- Minimum at $p_i = 1, \forall\, i = 1, 2, \ldots, N$

- Implementation supports parallel runs and provides analytical gradient and sparse Hessian
  - Simulation-based / HPC apps. would use algorithmic differentiation

- TAO can compute sensitivities via finite-differencing when analytical derivatives are not available
  - Convenient for prototyping or debugging
  - Computationally expensive for large optimization problems or expensive objectives

- Hands-on Activities:
  https://xsdk-project.github.io/MathPackagesTraining2020/lessons/multidim_rosenbrock_tao/

Take Away Messages

- PETSc/TAO offers parallel optimization algorithms for large-scale problems.
- Efficient gradients are needed for best results (e.g., algorithmic differentiation).
- Second-order methods don't always achieve faster/better solutions.
- When apps can only provide function evaluations, PETSc/TAO can automatically compute gradients via finite differencing.
- PETSc/TAO can also use finite differencing to validate application-provided gradients and Hessians.

Acknowledgements

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

Offline Questions: adener@anl.gov

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.