

# Numerical Optimization Using PETSc/TAO

Presented to  
**ATPESC 2022 Participants**

**Todd Munson**  
Mathematics and Computer Science Division  
Argonne National Laboratory

08/06/2024



## ATPESC Numerical Software Track



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  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ 
  - e.g.: lift, drag, max stress, total energy, error norms, etc.

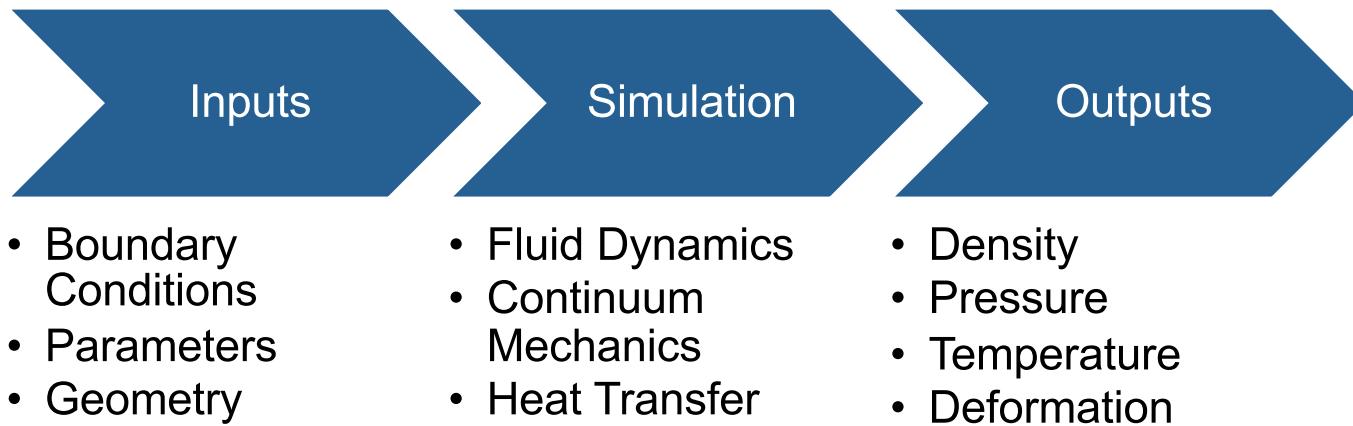
# What is optimization?

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

- 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

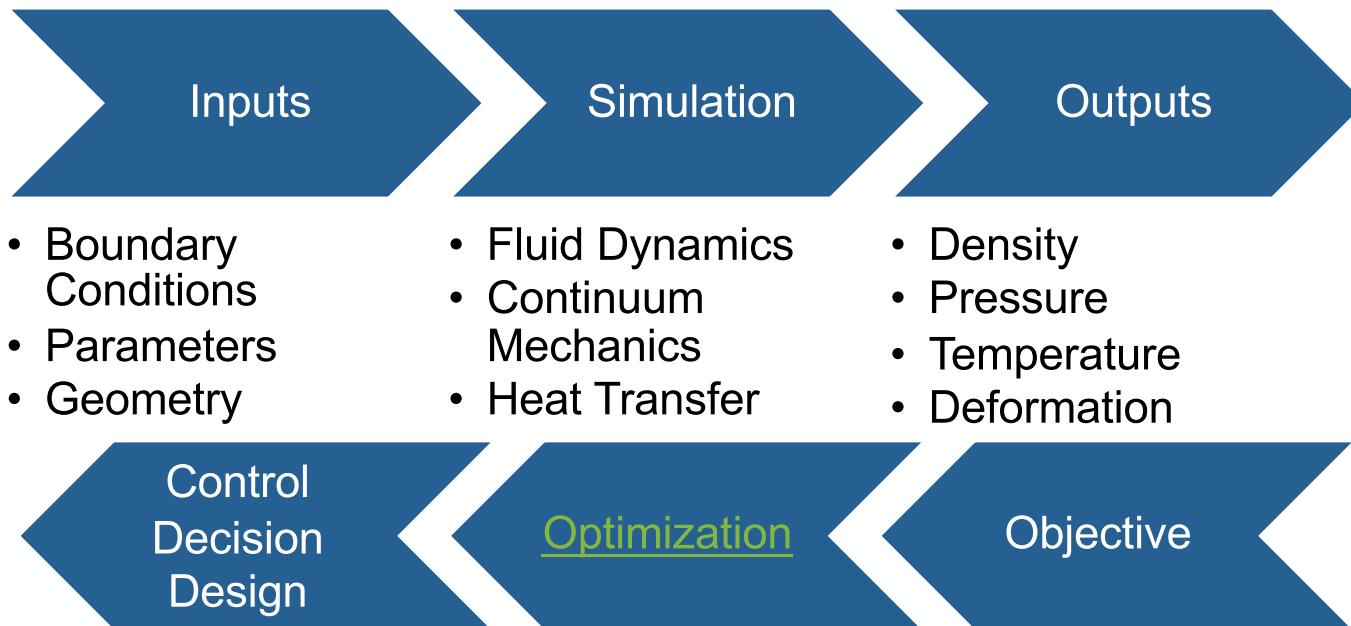
# Why do we care?

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



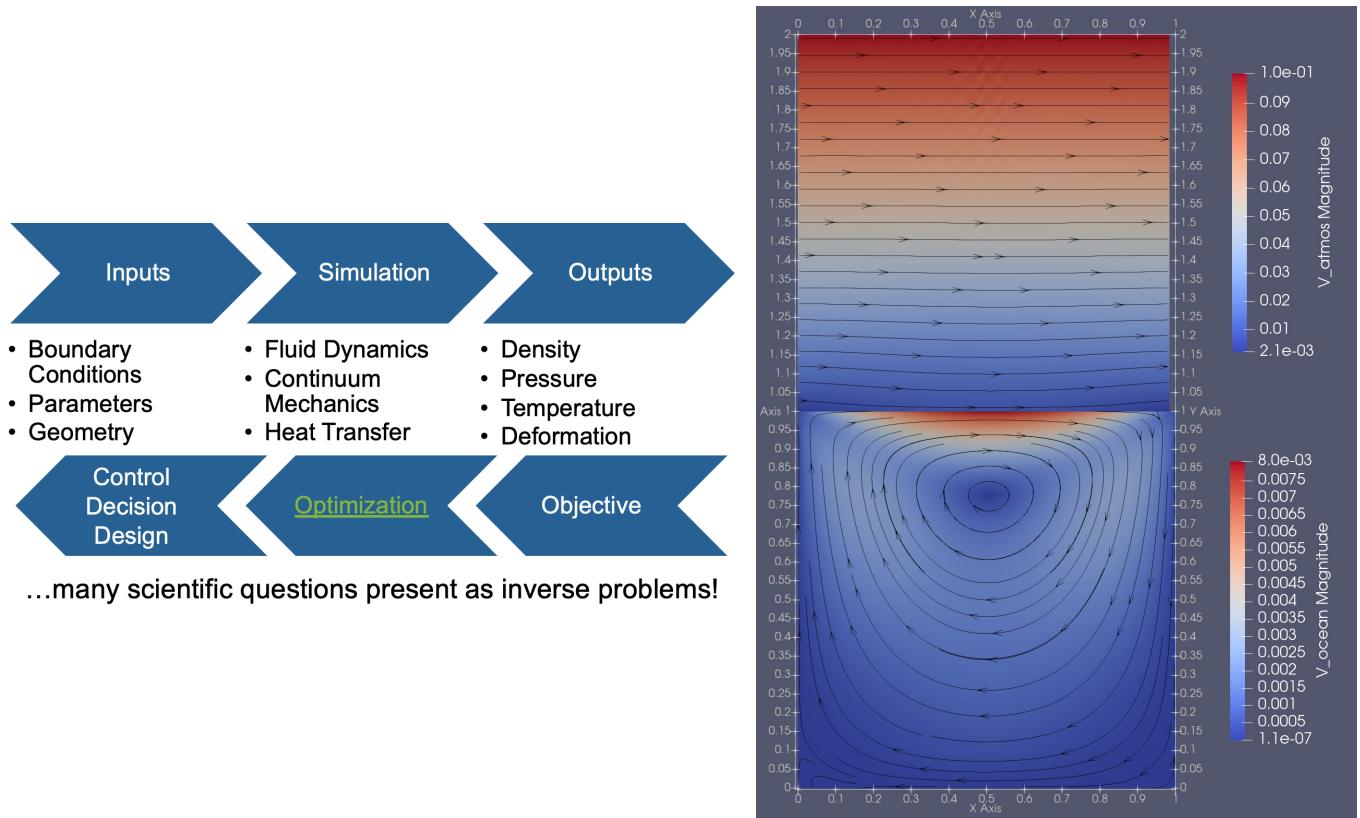
# Why do we care?

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



...many scientific questions present as inverse problems!

# Why do we care?



# Outline

- Introduction to Gradient-Based Optimization
  - Sequential Quadratic Programming
  - Sensitivity Analysis
- Introduction to TAO
  - Sample main program
  - User/problem callback function
- Hands-on Examples: Rosenbrock Equation
  - 2-dimensional unconstrained
  - Multidimensional unconstrained
  - 2-dimensional with general constraints

# 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., maximums) also satisfy  $\nabla_p f(p) = 0$

# Sequential Quadratic Programming

```
for k=0,1,2,... do
    mind fk + dTgk + 0.5dTHkd
    minα (α) = f(pk + αd)
    pk+1 pk + αd
end for
```

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

- 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

```
for k=0,1,2,... do
    mind fk + dTgk + 0.5dTHkd
    minα (α) = f(pk + αd)
    pk+1 pk + αd
end for
```

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

- 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 the secant condition
  - **Conjugate Gradient:**  $d_k = -g_k + \beta_k d_{k-1}$  with  $\beta$  defining different CG updates
  - **Gradient Descent:**  $d = -g_k$ , replace Hessian with identity

# PDE-Constrained Optimization

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

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

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

## Full-Space Formulation

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

## Reduced-Space Formulation

- State variables are implicit functions of parameters

# 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://petsc.org>)
  - Similar packages:
    - Rapid Optimization Library (<https://trilinos.github.io/rol.html>)
    - HiOP (<https://github.com/LLNL/hiop>)
- |       |    |     |
|-------|----|-----|
| PETSc |    |     |
| TAO   |    |     |
| TS    |    |     |
| SNES  |    |     |
| KSP   | PC |     |
| DM    |    |     |
| Vec   |    | Mat |

# 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, TAQBQNLS);           /* bound-constrained quasi-Newton */
TaoSetSolution(tao, P);
TaoSetObjective(tao, FormFunction, &user);
TaoSetGradient(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, TAQBQNLS);           /* bound-constrained quasi-Newton */
TaoSetSolution(tao, P);
TaoSetObjective(tao, FormFunction, &user);
TaoSetGradient(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  $G = \nabla_p f$  at given  $p$
- **(ADVANCED) Second-order Methods:**
  - Compute  $H = \nabla_p^2 f$  at given  $p$
  - Use `TaoSetHessian()` interface
- **(ADVANCED) Constraints:**
  - Set bound constraints  $p_l \leq p \leq p_u$
  - Define nonlinear constraints  $c_e(p) = 0, c_i(p) \geq 0$

# TAO: User Function

- **Objective evaluation:**

- Compute  $f(p)$  at given  $p$

- **Sensitivity analysis:**

- Compute  $G = \nabla_p f$  at given  $p$

- **(ADVANCED) Second-order Methods:**

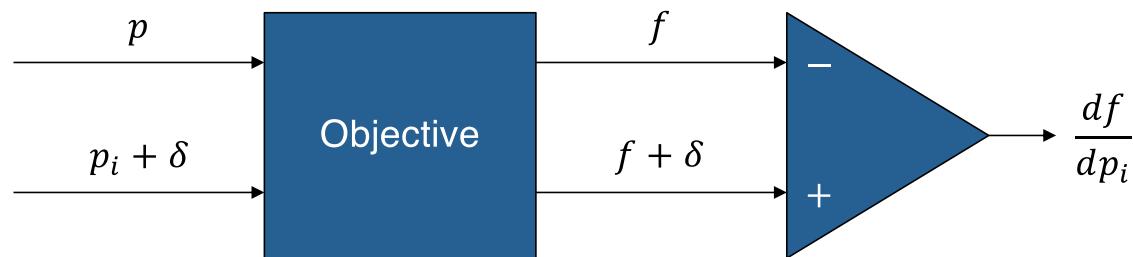
- Compute  $H = \nabla_p^2 f$  at given  $p$
  - Use `TaoSetHessian()` interface

- **(ADVANCED) Constraints:**

- Set bound constraints  $p_l \leq p \leq p_u$
  - Define nonlinear constraints  $c_e(p) = 0, c_i(p) \geq 0$

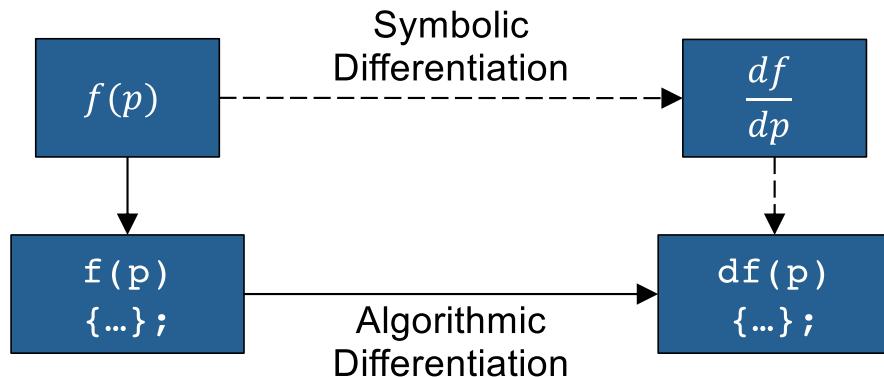
- Necessary for gradient-based optimization algorithms
- Types:
  - Numerical
  - Analytical

# 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** – hand-derived gradient with direct code implementation
- **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
- Some popular AD tools:
  - ADIC (ANSI C)
  - ADIFOR (Fortran77/Fortran95)
  - OpenAD (Fortran77/Fortran95/C/C++)
  - Sacado (C/C++)
  - ForwardDiff.jl (Julia)
  - JAX (Python)

# TAO: Bound Constraints

- What if we wanted to restrict the solution variables?

minimize  $f(x)$

subject to  $x_l \leq x \leq x_u$

```
VecDuplicate(x, &XL);
VecSet(XL, PETSC_NINFINITY);
VecDuplicate(x, &XU);
VecSet(XU, 0.0);
TaoSetVariableBounds(tao, XL, XU);
```

- Must use bound-constrained TAO algorithms
  - TAOBNLS: Bounded Newton Line-Search
  - TAOBNTR: Bounded Newton Trust Region
  - TAOBQNLS: Bounded quasi-Newton Line-Search
  - TAOBNCG: Bounded Nonlinear Conjugate Gradient

# TAO: General Nonlinear Constraints

- Incorporate all constraint types

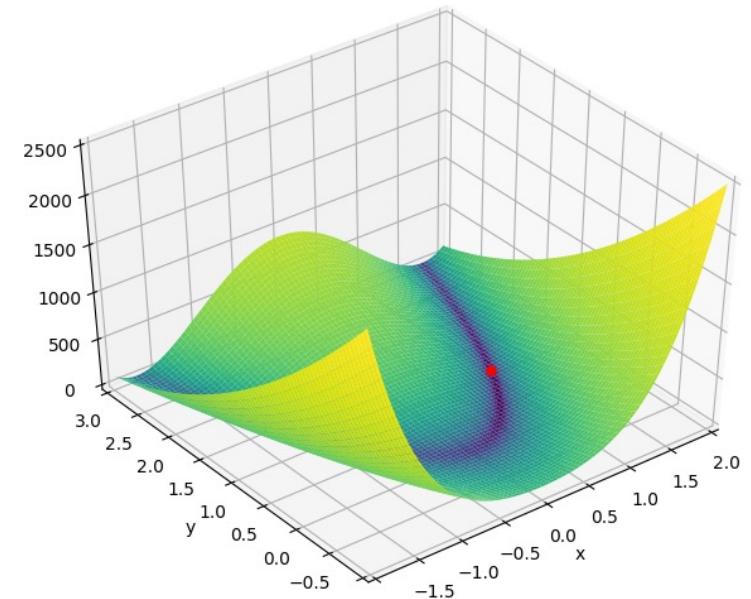
$$\begin{aligned} & \underset{x}{\text{minimize}} && f(x) \\ & \text{subject to} && c_e(x) = 0 \\ & && c_i(x) \leq 0 \\ & && x_l \leq x \leq x_u \end{aligned}$$

- Must use TAOALMM solver
  - Augmented Lagrangian method w/ interior-point formulation for inequality constraints
- Define constraints with user call-backs
  - `FormEqualityConstraints(Tao,Vec,Vec,void*)` → `TaoSetEqualityConstraintsRoutine()`
  - `FormEqualityJacobian(Tao,Vec,Mat,Mat,void*)` → `TaoSetJacobianEqualityRoutine()`
  - `FormInequalityConstraints(Tao,Vec,Vec,void*)` → `TaoSetInequalityConstraintsRoutine()`
  - `FormInequalityJacobian(Tao,Vec,Mat,Mat,void*)` → `TaoSetJacobianInequalityRoutine()`

# Hands-on Example: 2-dimensional Rosenbrock

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

- 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



# Hands-on Example: Multidimensional Rosenbrock

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

- Global minimum at  $p_i = 1, \forall i = 1, 2, \dots, N$
- 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:

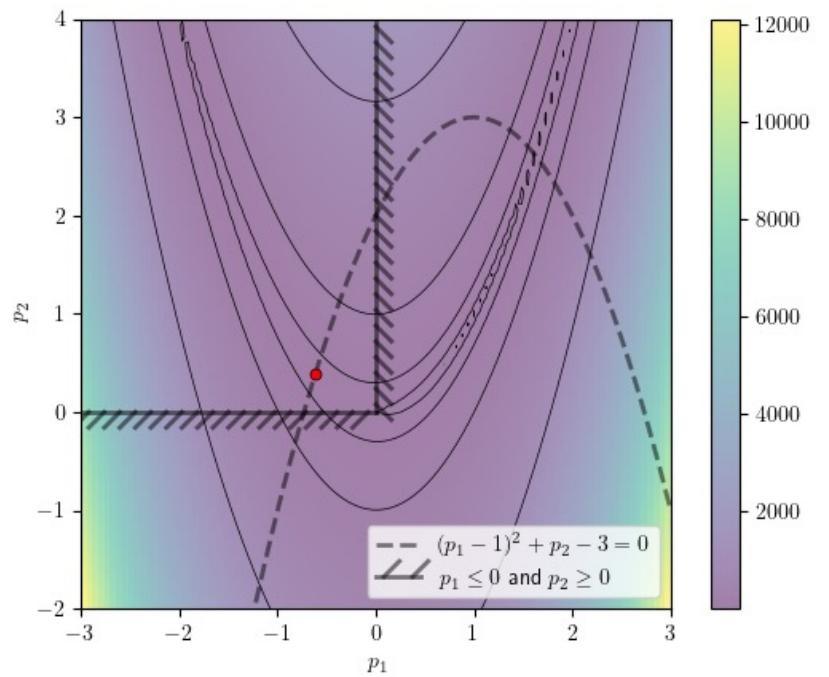
[https://xSDK-project.github.io/MathPackagesTraining2024/lessons/numerical\\_optimization\\_tao/](https://xSDK-project.github.io/MathPackagesTraining2024/lessons/numerical_optimization_tao/)

# Hands-on Example: 2D Rosenbrock w/ Constraints

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

$$\begin{aligned} \text{subject to} \quad & (p_1 - 1)^2 + p_2 - 3 = 0 \\ & p_1 \leq 0 \\ & p_2 \geq 0 \end{aligned}$$

- Bound-constrained global minimum at  $f(0,0) = 1$
- Equality-constraints have two local minima
  - $f(-0.62, 0.38) = 2.62$
  - $f(1.62, 2.62) = 0.38$
- Combined constraints yield global minimum at  $f(-0.62, 0.38) = 2.62$
- Constraints are not valid for the multidimensional Rosenbrock problem



## Take Away Messages

- PETSc/TAO offers parallel optimization algorithms for large-scale problems.
- Efficient gradients are needed for best results (e.g., algorithmic differentiation), and second-order methods don't always achieve faster/better solutions.
- PETSc/TAO can automatically compute gradients via finite differencing.
- PETSc/TAO can incorporate bound, equality and inequality constraints into the solution.
- Most scientific problems are nonlinear and nonconvex... starting point matters!

# Acknowledgements

PETSc/TAO Documentation: <https://petsc.org>

GitLab Repo: <https://gitlab.com/petsc/petsc>

Offline questions: [tmunson@anl.gov](mailto:tmunson@anl.gov)

Need help with your PETSc/TAO application? [petsc-users@mcs.anl.gov](mailto:petsc-users@mcs.anl.gov)