SAND2006-6911P

# Sensitivities and Optimization:

# Going Beyond the Forward Solve

# (to Enable More Predictive Simulations)

**Roscoe A. Bartlett**
**Department of Optimization & Uncertainty Estimation**

**Sandia National Laboratories**

**Trilinos Users Group Meeting, November 8th, 2006**

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

# Outline

- Mathematical overview of sensitivities and optimization
- Minimally invasive optimization algorithm for MOOCHO
- ModelEvaluator software
- Wrap it up

# Outline

- Mathematical overview of sensitivities and optimization
- Minimally invasive optimization algorithm for MOOCHO
- ModelEvaluator software
- Wrap it up

# Why Sensitivities and Optimization?

## The Standard Steady-State Forward Simulation Problem

For a given set of input parameters $p \in \mathbf{R}^{n_p}$, solve the square state equations

$$f(x,p) = 0$$

for the state variables $x \in \mathbf{R}^{x_x}$ then compute observation(s) $g(x)$.

## Example applications

- Discretized PDEs (e.g. finite element, finite volume, discontinous Gelerkin, finite difference, …)
- Network problems (e.g. circuit simulation, power grids)
- …

### Why is a forward solver is not enough?

- A forward solve *p* → *g(x(p),p)* can only give point-wise information, it can't tell you what you ultimately want to know:
  - How to a characterize the error in my model so that it can be improved? → Error estimation
  - What is the uncertainty in *x* given uncertainty in *p* ? → QMU
  - What is the "best" value of *p* so that my model *f(x,p)=0* fits exp. data? → Param. Estimation
  - What is the "best" value for *p* to achieve some goal? → Optimization

## What are some of the tools that we need to answer these higher questions?

- Sensitivities and Optimziation!

Sandia National Laboratories

# Steady-State Simulation-Constrained Sensitivities

## Steady-State Simulation-Constrained Response

Compute $g(x,p) \in \mathbf{R}^{n_x} \times \mathbf{R}^{n_p} \rightarrow \mathbf{R}^{n_g}$

such that $f(x,p) = 0$ (where $f(x,p) \in \mathbf{R}^{n_x} \times \mathbf{R}^{n_p} \rightarrow \mathbf{R}^{n_x}$ )

Nonlinear elimination: $f(x,p) = 0 \longrightarrow p \rightarrow x(p) \longrightarrow$ Reduced Response Function: $p \rightarrow \hat{g}(p) = g(x(p),p)$

## Steady-State Sensitivities

State Sensitivity: $$\frac{\partial x}{\partial p} = -\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}$$

Well suited for Newton Methods

Reduced Response Function Sensitivity: $$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial g}{\partial p}$$

## Forward (Direct) vs. Adjoint Sensitivities

Complexity

Forward (Direct) Sensitivity Method: $$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial x}\left(-\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}\right) + \frac{\partial g}{\partial p}$$ $O(n_p)$

Adjoint Sensitivity Method: $$\frac{\partial \hat{g}}{\partial p}^T = \frac{\partial f}{\partial p}^T\left(-\frac{\partial f}{\partial x}^{-T}\frac{\partial g}{\partial x}^T\right) + \frac{\partial g}{\partial p}^T$$ $O(n_g)$

Uses for Sensitivities: Derivative-based optimization, UQ, error estimation etc …

Sandia National Laboratories

# Steady-State Simulation-Constrained Optimization

## Basic Steady-State Simulation-Constrained Optimization Problem:

Find $x \in \mathbf{R}^{n_x}$ and $p \in \mathbf{R}^{n_p}$ that:
minimizes $g(x,p)$
such that $f(x,p) = 0$

## Basic example optimization formations

- Parameter estimation / data reconciliation
- Optimal design
- Optimal control
- …

**Trilinos Optimization Packages**
- MOOCHO (R. Bartlett)
- Aristos (D. Ridzal)

Define Lagrangian: $L(x,p,\lambda) = g(x,p) + \lambda^T f(x,p)$

## Optimality conditions

State equation: $$\frac{\partial L}{\partial \lambda}^T = f(x,p) = 0$$

Adjoint equation: $$\frac{\partial L}{\partial x}^T = \frac{\partial g}{\partial x}^T + \frac{\partial f}{\partial x}^T \lambda = 0$$

Gradient equation: $$\frac{\partial L}{\partial p}^T = \frac{\partial g}{\partial p}^T + \frac{\partial f}{\partial p}^T \lambda = 0$$

$$\frac{\partial \hat{g}}{\partial p}^T = -\frac{\partial f}{\partial p}^T \frac{\partial f}{\partial x}^{-T} \frac{\partial g}{\partial x}^T + \frac{\partial g}{\partial p}^T$$

Reduced sensitivity!

# Simulation-Constrained Optimization Methods

Basic Steady-State Simulation-Constrained Optimization Problem:

Find $x \in \mathbf{R}^{n_x}$ and $p \in \mathbf{R}^{n_p}$ that:
minimizes $g(x,p)$
such that $f(x,p) = 0$

Two broad approaches for solving optimization problems

- Decoupled approach (simulation constraints always satisfied): **DAKOTA**

Find $p \in \mathbf{R}^{n_p}$ that:
minimizes $\hat{g}(p) = g(x(p),p)$

Optimization method never "sees" the state space!

- Coupled approach (converges optimality and feasibility together): **MOOCHO, Aristos**

Find $x \in \mathbf{R}^{n_x}$ and $p \in \mathbf{R}^{n_p}$ that:
minimizes $g(x,p)$
such that $f(x,p) = 0$

- Optimization method deals with the (parallel) state space and the parameter space together!
- Requires special globalization methods to converge to a minimum!

Sandia National Laboratories

# Full-Newton Coupled Optimization Methods

Optimality conditions

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial x}^T \\ \frac{\partial L}{\partial p}^T \\ \frac{\partial L}{\partial \lambda} \end{bmatrix} = \begin{bmatrix} \frac{\partial g}{\partial x}^T + \frac{\partial L}{\partial x}^T \lambda \\ \frac{\partial g}{\partial p}^T + \frac{\partial L}{\partial p}^T \lambda \\ f(x,p) \end{bmatrix} = 0$$

A set of three coupled nonlinear equations!

Solve using Newton's method?

Full-Newton Coupled Optimization Methods (The Gold Standard)

$$\begin{bmatrix} \frac{\partial^2 L}{\partial x^2} & \frac{\partial^2 L}{\partial x \partial p} & \frac{\partial f}{\partial x}^T \\ \frac{\partial^2 L}{\partial x \partial p}^T & \frac{\partial^2 L}{\partial p^2} & \frac{\partial f}{\partial p}^T \\ \frac{\partial f}{\partial x} & \frac{\partial f}{\partial p} & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta p \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \frac{\partial g}{\partial x}^T + \frac{\partial L}{\partial x}^T \lambda \\ \frac{\partial g}{\partial p}^T + \frac{\partial L}{\partial p}^T \lambda \\ f(x,p) \end{bmatrix}$$

Also know as a full-space successive quadratic programming (SQP) method!

**Aristos** (D. Ridzal)

- Results in fast local quadratic (Newton) convergence
- Global convergence to a minimum requires special "globalization" methods
- Requires second derivatives (i.e. Hessians)
- Requires solution of large symmetric indefinite systems
- Hard to exploit forward-solve capabilities of an application

Sandia National Laboratories

# Reduced-Space Coupled Optimization Methods (i.e. MOOCHO)

## Basic Steady-State Simulation-Constrained Optimization Problem

Find $x \in \mathbf{R}^{n_x}$ and $p \in \mathbf{R}^{n_p}$ that:
minimizes $g(x,p)$
such that $f(x,p) = 0$

## Basic (line-search-based) reduced-space optimization algorithm

1. Initialization: Choose tolerances $\eta_f, \eta_g \in \mathbf{R}$ and the initial guess $x_0 \in \mathbf{R}^{n_x}$ and $p_0 \in \mathbf{R}^{n_p}$, set $k = 0$
2. Model/sensitivity evaluation: Compute the reduced derivative $\partial\hat{g}/\partial p$ and the residual $f$ at $(x_k, p_k)$
3. Convergence check: If $||\partial\hat{g}/\partial p|| \le \eta_g$ and $||f|| \le \eta_f$ then stop, solution found!
4. Step computation:
   (a) Feasiblity step: Compute Newton step $\Delta x_N = (\partial f/\partial x)^{-1} f$ at $(x_k, p_k)$
   (b) Optimality step: Compute $\Delta p \in \mathbf{R}^{n_p}$ s.t. $(\partial\hat{g}/\partial p)\Delta p < 0$
5. Globalization: Find step length $\alpha$ that insures progress to the solution
6. Update the estimate of the solution:
   $x_{k+1} = x_k + \alpha\,(\Delta x_N + (\partial x/\partial p)\Delta p)$
   $p_{k+1} = p_k + \alpha\,\Delta p$
   $k = k + 1$
   goto step 2

$$\frac{\partial x}{\partial p} = -\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}$$

$$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial g}{\partial p}$$

- Strongly leverages the capabilities of applications that can be used with a Newton-based forward solver
- Direct sensitivities
  - $n_p+1$ solves with $\partial f/\partial x$
- Adjoint sensitivities
  - *2* solves with $\partial f/\partial x$
  - 1 solve with $(\partial f/\partial x)^T$
- Quasi-Newton methods typically used to approximate reduced Hessian (second derivatives) $B$ and to compute:

  $\Delta p = -B^{-1} (\partial\hat{g}/\partial p)^T$

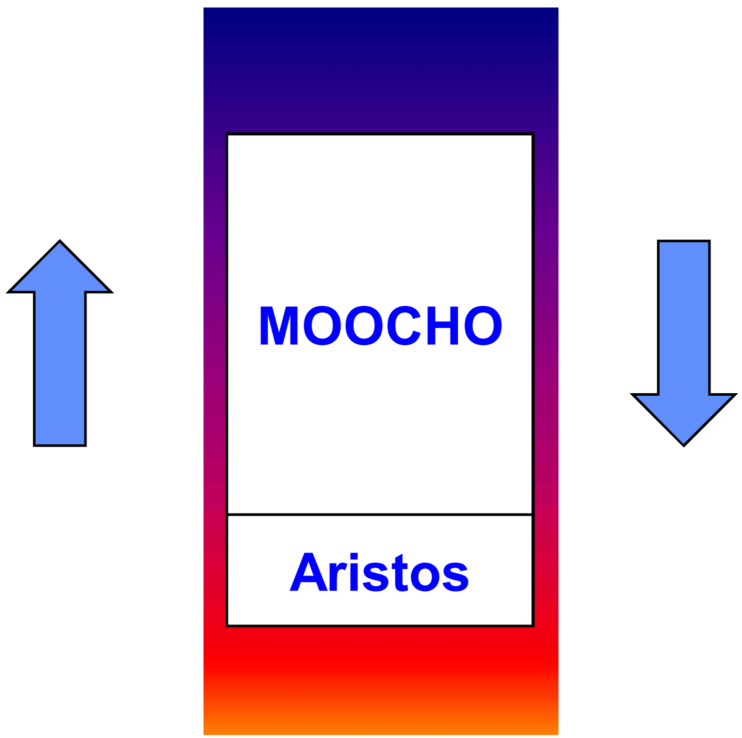
- MOOCHO implements this class and related classes of algorithms!

Sandia National Laboratories

# A Spectrum of Optimization Methods form Decoupled to Coupled

- Decreased impact to existing app code
- Ease of interfacing

**Fully Decoupled Optimization Method**




**Fully Coupled, Newton Optimization Method**

- Better scalability to large parameter spaces
- More accurate solutions
- Less computer time

# General Inequality Simulation-Constrained Optimization

## General Steady-State Simulation-Constrained Optimization Problem with Inequalities:

Find $x \in \mathbf{R}^{n_x}$ and $p \in \mathbf{R}^{n_p}$ that:
minimizes $g_0(x,p)$
such that:

$$f(x,p) = 0$$
$$g_1(x,p) = 0$$
$$g_{L,2} \le g_2(x,p) \le g_{U,2}$$
$$x_L \le x \le x_U$$
$$p_L \le p \le p_U$$

MOOCHO allows inequality constraints using active-set methods (see QPSchur)!

## Example optimization formations

- Parameter estimation / data reconciliation
- Optimal design
- Optimal control
- …

## Issues associated with handling of inequalities

- Inequalities allow a modeling capability not possible with just equalities
- Allows reformulation of some non-differentiable optimization problems
- Two broad classes of optimization algorithms for handling inequalities
  - Active-set methods (adds and removed inequalities from "working set")
  - Interior-point methods (enforces inequalities using "barrier term")
- Tradeoffs between direct and adjoint sensitivity methods not so obvious anymore

# Transient Sensitivities and Optimization

| | |
|---|---|
| Explicit ODE Forward Sensitivities: | Find $\frac{\partial x}{\partial p}(t)$ such that: $\dot{x} = f(x,p,t) = 0, t \in [0,T]$, $x(0) = x_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$ |
| DAE/Implicit ODE Forward Sensitivities: | Find $\frac{\partial x}{\partial p}(t)$ such that: $f(\dot{x}(t), x(t), p, t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$<br>Find $p \in \mathbf{R}^m$ that minimizes $g(p)$ |
| ODE Constrained Optimization: | Find $x(t) \in \mathbf{R}^n$ in $t \in [0,T]$ and $p \in \mathbf{R}^m$ that:<br>minimizes $\int_0^T g(x(t),p)$<br>such that $\dot{x} = f(x(t),p,t) = 0$, on $t \in [0,T]$<br>where $x(0) = x_0$ |

Issues associated with transient sensitivities and optimization

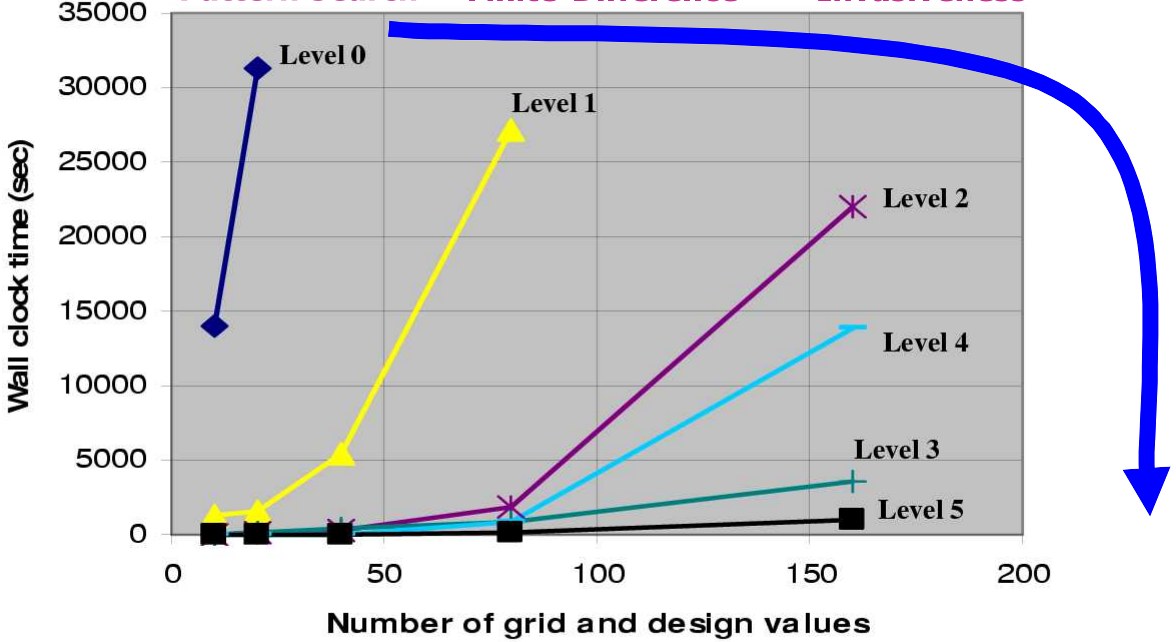
- If sufficient storage is available, then discretization in time yields a **BIG** steady-state problem
  - ⇒ 4D approach!
- If sufficient storate is not available then time integration methods must be used to eliminate transient equations and state varaibles
  - Direct transient sensitivity methods scale as *$O(n_p)$*
  - Adjoint transient sensitivity methods scale as *$O(n_g)$* but require storage/recomputation of *$x(t)$*
- Model/Application requirements are similar for steady-state sensitivities and optimization

I am not going to say anything more about transient sensitivities or optimization!

# Outline

- Mathematical overview of sensitivities and optimization
- Minimally invasive optimization algorithm for MOOCHO
- ModelEvaluator software
- Wrap it up

# Motivation for Invasive Coupled Optimization




**Increasing Levels of Coupling and Derivative and Solve Capabilities**

**Level 2 = Decoupled forward sens.**
**Level 3 = Coupled forward sens.**
**Level 4 = Decoupled adjoint sens.**
**Level 5 = Coupled adjoint sens.**
**Level 6 = Coupled full-Newton**

**Large Scale Non-Linear Programming for PDE Constrained Optimization**, van Bloemen Waanders, B., Bartlett, R., Long, K., Boggs, P., and Salinger, A. Sandia Technical Report SAND2002-3198, October 2002

**Key Point**

For many/some optimization problems, intrusive coupled optimization methods can be much more computationally efficient and more robust than the decoupled approach

**But:**

- It is hard to get our "foot in the door" with production codes
- It is hard to keep a "door stop" in place once we are in … Because …

Sandia National Laboratories

# Some Challenges to Incorporation of Invasive Optimization

- Lack of Software Infrastructure
  - Linear algebra and linear solvers not supporting optimization requirements
  - Application structure not flexible (i.e. only supports a narrow mode to solve the forward problem)
- Lack of software maintenance
  - Optimization support is not tightly integrated with forward solve code and is not maintained over time.
- Lack of derivative support
  - Lack of model smoothness
  - No optimization variables derivatives
  - Lack of transient derivatives

Where I am Involved

**Thyra**

**MOOCHO**

**Model Evaluator**

**Automatic Differentation (AD)**

**Rythmos**

**Key Point**

We need a strategy to reduce the threshold for getting invasive optimization into codes and for keeping the capability once it is there => Software and Algorithms

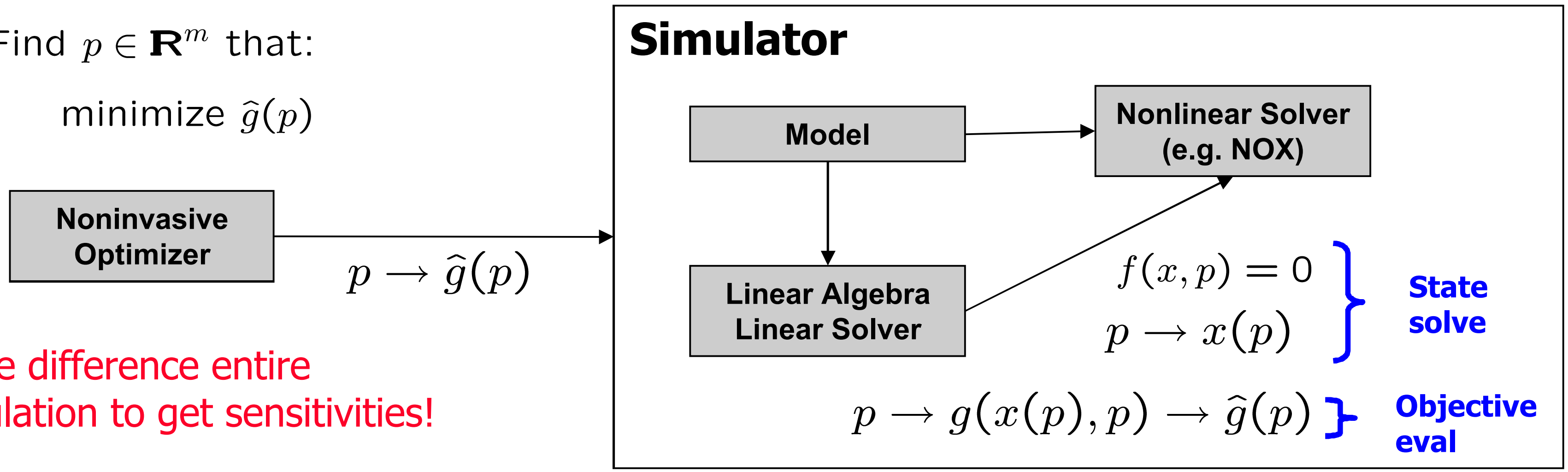
Sandia National Laboratories

# Minimally Invasive Gradient-Based Optimization

**Decoupled Optimization**: Assume there is no optimization capability in the "Simulator"

Find $p \in \mathbf{R}^m$ that:

minimize $\hat{g}(p)$

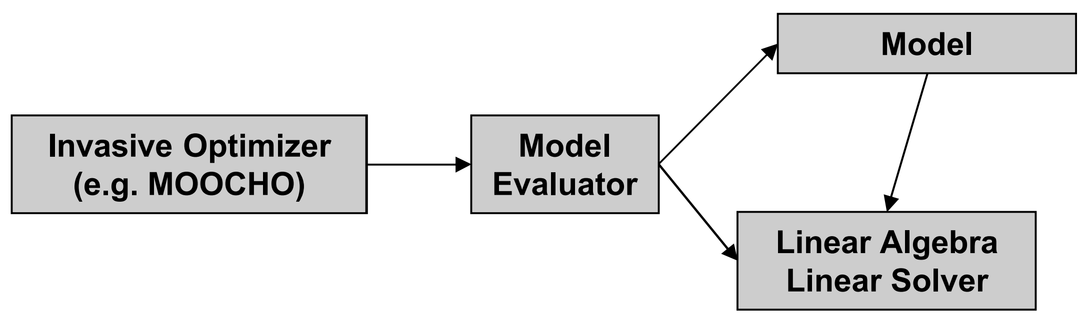



Finite difference entire simulation to get sensitivities!

**Coupled Optimization**: Simulator broken up and some pieces are given over to optimizer

Find $x \in \mathbf{R}^n$ and $p \in \mathbf{R}^m$ that:

minimizes $g(x,p)$
such that
$f(x,p) = 0$




**Question**: How can we break the a simulator open to begin using coupled optimization methods and how can our algorithms exploit any capabilities that the simulator can provide?

# Minimally Invasive Direct Sensitivity MOOCHO

## Basic Simulation-Constrained Optimization Problem

Find $x \in \mathbf{R}^n$ and $p \in \mathbf{R}^m$ that:

minimize $g(x,p)$
such that
$f(x,p) = 0$

**Defines the state simulator and direct sensitivities**

$p \longrightarrow x(p)$

$$\frac{\partial x}{\partial p} = -\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}$$

**Reduced Obj. Function**

$p \longrightarrow \hat{g}(p)$

## Minimal Requirements for decoupled Newton simulation-constrained optimization

– Residual Eval: $(x,p) \rightarrow f$

– Jacobian Eval: $(x,p) \rightarrow \frac{\partial f}{\partial x}$

– Objective Eval: $(x,p) \rightarrow g$

**Linear Solver** } **State solve with NOX/LOCA**

} **Decoupled Opt.**

} **Minimally Invasive Direct Sensitivity MOOCHO**

## Derivatives desired but not required

– Residual opt. deriv: $(x,p) \rightarrow \frac{\partial f}{\partial p}$

– Objective state deriv: $(x,p) \rightarrow \frac{\partial g}{\partial x}$

– Objective opt. deriv: $(x,p) \rightarrow \frac{\partial g}{\partial p}$

## Approximate using O($n_p$) directional finite differences!

$$\frac{\partial f}{\partial p_i} \approx \frac{f(x, p + \delta e_i) - f(x,p)}{\delta}$$

$$\frac{\partial \hat{g}}{\partial p_i} \approx \frac{g\left(x + \delta\frac{\partial x}{\partial p_i}, p + \delta e_i\right) - g(x,p)}{\delta}$$

# Scalable Optimization Test Problem

Example: Parallel, Finite-Element, 2D, Diffusion + Reaction (GL) Model

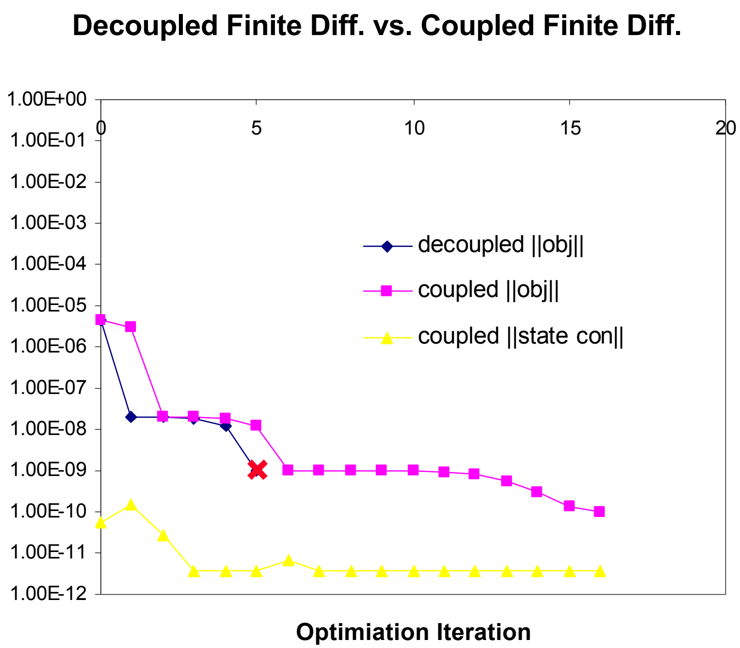
$$\begin{array}{lll} \text{min} & \frac{1}{2}\int_\Omega (x(y) - x^*(y))^2 dy & \\ \text{s.t.} & \nabla^2 x + \alpha(x - x^3) = r(y) & y \in \Omega \\ & \frac{\partial x(y)}{\partial n} = q(p, y) & y \in \partial\Omega \end{array} \quad \Rightarrow \quad \begin{array}{ll} \text{min} & g(x,p) \\ \text{s.t.} & f(x,p) = 0 \end{array}$$

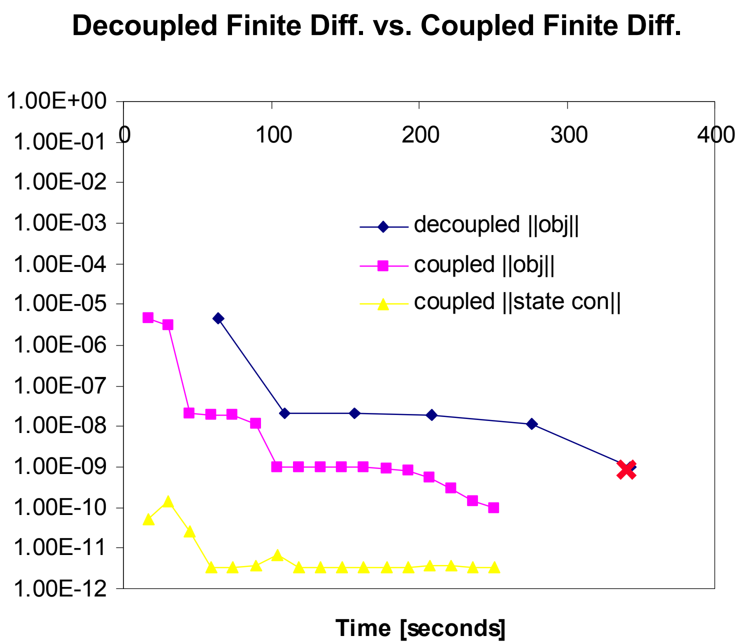
- State PDE: Scalar Ginzburg-Landau equations (based on Denis Ridzal's (1414) code)
- Discretization:
  - Second-oder FE on triangles
  - $n_x$ = 110,011 state variables and equations
- Optimization variables:
  - Sine series basis
  - $n_p$ = 8 optimization variables
  - Note: df/dp is constant in this problem!!!
- Iterative Linear Solver : ILU (Ifpack), (GMRES) AztecOO

**Key Points**

- Simple physics but leads to very nonlinear state equations
- Inverse optimization problem is very ill posed in many instances

# Results: Decoupled vs. Coupled, Finite Differences



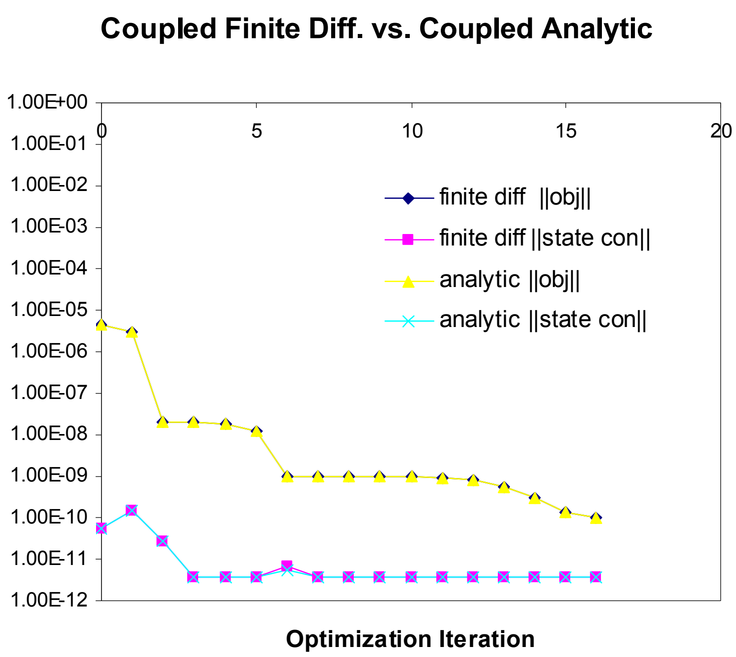



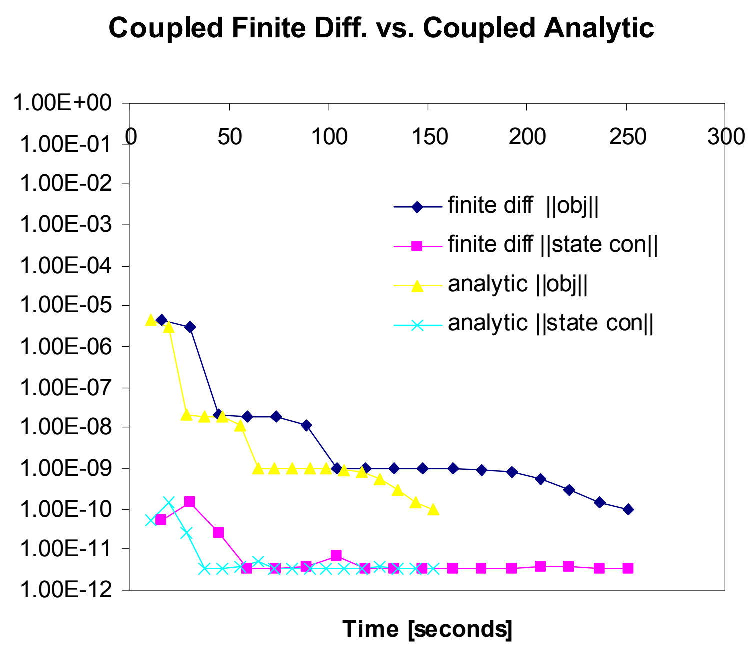

## Key Points

- Finite differencing the underlying functions is much more efficient than finite differencing entire simulation!
- Finite differencing the underlying functions is more accurate!
- Coupled approach requires (almost) no extra application requirements!

Sandia National Laboratories

# Results: Coupled Finite Diff. vs. Coupled Analytic




**Key Points**

- Analytic derivatives are usually not faster
- Analytic derivatives often much more accurate

Sandia National Laboratories

# Outline

- Mathematical overview of sensitivities and optimization
- Minimally invasive optimization algorithm for MOOCHO
- ModelEvaluator software
- Wrap it up

Sandia National Laboratories

# Overview of Nonlinear Model Evaluator Interface

Motivation: An interface for nonlinear problems is needed that will support a variety of different types of problems

- Nonlinear equations (and senstitivities)
- Stability analysis and continuation
- Explicit ODEs (and sensitivities)
- DAEs and implicit ODEs (and sensitivities)
- Unconstrained optimization
- Constrained optimization
- Uncertainty quantification
- …

as well as different combinations of problem types such as:

- Uncertainty in transient simulations
- Stability of an optimum under uncertainty of a transient problem

**Key Point**

The number of combinations of different problem types is large and trying to statically type all of the combinations is not realistic

Approach: Develop a single, scalable interface to address all of these problems

- (Some) Input arguments:
  - State and differential state: $x \in \mathcal{X}$ and $\dot{x} = \frac{dx}{dt} \in \mathcal{X}$
  - Parameter sub-vectors: $p_l \in \mathcal{P}_l$ for $l = 1 \ldots N_p$
  - Time (differential): $t \in \mathbf{R}$
- (Some) Output functions:
  - State function: $(\dot{x}, x, \{p_l\}, t) \Rightarrow f \in \mathcal{F}$
  - Auxiliary response functions: $(\dot{x}, x, \{p_l\}, t) \Rightarrow g_j \in \mathcal{G}_j$, for $j = 1 \ldots N_g$
  - State/state derivative operator (LinearOpWithSolve): $(\dot{x}, x, \{p_l\}, t) \Rightarrow W = \alpha \frac{\partial f}{\partial \dot{x}} + \beta \frac{\partial f}{\partial x}$

**Key Point**

All inputs and outputs are optional and the model evaluator object itself decides which ones are accepted.

# Some Examples of Supported Nonlinear Problem Types

| | |
|---|---|
| Nonlinear equations: | Solve $f(x) = 0$ for $x \in \mathbf{R}^n$ |
| Stability analysis: | For $f(x,p) = 0$ find space $p \in \mathcal{P}$ such that $\frac{\partial f}{\partial x}$ is singular |
| Explicit ODEs: | Solve $\dot{x} = f(x,t) = 0, t \in [0,T]$, $x(0) = x_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$ |
| DAEs/Implicit ODEs: | Solve $f(\dot{x}(t), x(t), t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$ for $x(t) \in \mathbf{R}^n, t \in [0,T]$ |
| Explicit ODE Forward Sensitivities: | Find $\frac{\partial x}{\partial p}(t)$ such that: $\dot{x} = f(x,p,t) = 0, t \in [0,T]$, $x(0) = x_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$ |
| DAE/Implicit ODE Forward Sensitivities: | Find $\frac{\partial x}{\partial p}(t)$ such that: $f(\dot{x}(t), x(t), p, t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$ |
| Unconstrained Optimization: | Find $p \in \mathbf{R}^m$ that minimizes $g(p)$ |
| Constrained Optimization: | Find $x \in \mathbf{R}^n$ and $p \in \mathbf{R}^m$ that: minimizes $g(x,p)$ such that $f(x,p) = 0$ |
| ODE Constrained Optimization: | Find $x(t) \in \mathbf{R}^n$ in $t \in [0,T]$ and $p \in \mathbf{R}^m$ that: minimizes $\int_0^T g(x(t),p)$ such that $\dot{x} = f(x(t),p,t) = 0$, on $t \in [0,T]$ where $x(0) = x_0$ |

Sandia National Laboratories

# A More Advanced Optimization Example

## Equality and Inequality Constrained Optimization solved using Continuation:

Find $x \in \mathbf{R}^n$ and $p_0 \in \mathbf{R}^m$ that:

minimizes $g_0(x, p_0)$
such that

$$f(x, p_0, p_1) = 0$$
$$g_1(x, p_0, p_1) = 0$$
$$g_2^L \leq g_2(x, p_0, p_1) \leq g_2^U$$

using continuation parameters $p_1$

- $N_p$ = 2 parameter sub-vectors:
  - design $p_0$
  - continuation $p_1$
- $N_g$ = 3 response functions:
  - objective $g_0 \in \mathbf{R}^1$
  - auxiliary equalities $g_1$
  - auxiliary inequalities $g_2$

**Key Points**

- This is a very realistic problem that could be added to MOOCHO this year!
- We can't be developing a new interface for every one of these types of mixed problem formulations!

# Example : "Composite" Coupled (Multi-Physics) Models

## Forward Coupled Model:

$$\tilde{f}^1(\tilde{x}^1, \tilde{p}_1^1) = 0$$

$$\tilde{p}_1^2 = \tilde{x}^1$$

$$\tilde{f}^2(\tilde{x}^2, \tilde{p}_1^2) = 0$$

## "Composite" Forward Coupled Model:

$$f(x, p_1) = 0$$

where

$$x = \begin{bmatrix} \tilde{x}^1 \\ \tilde{x}^2 \end{bmatrix}$$

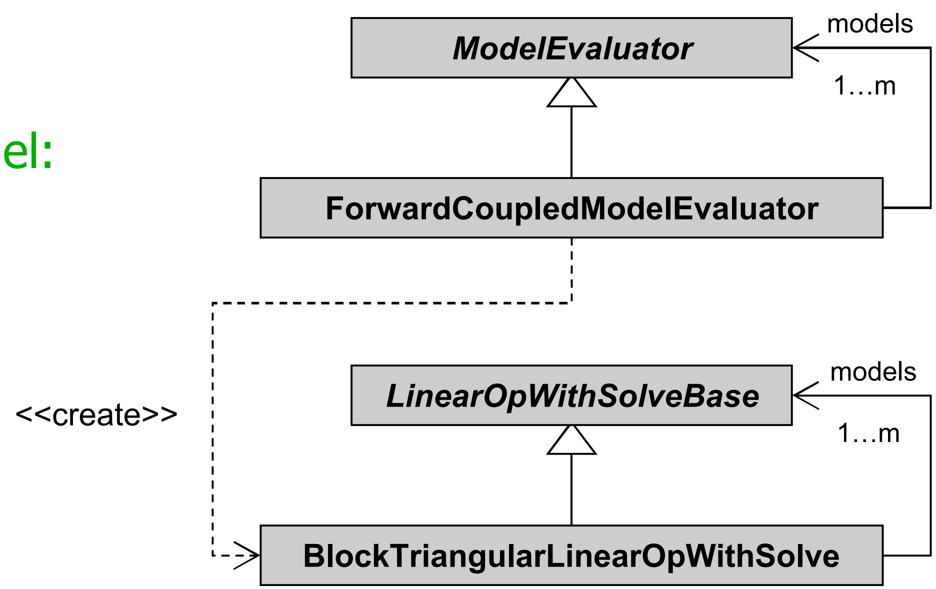
$$p_1 = \tilde{p}_1^1$$

$$f(x, p_1) = \begin{bmatrix} \tilde{f}^1(\tilde{x}^1, \tilde{p}_1^1) \\ \tilde{f}^2(\tilde{x}^2, \tilde{p}_1^2 = \tilde{x}^1) \end{bmatrix}$$

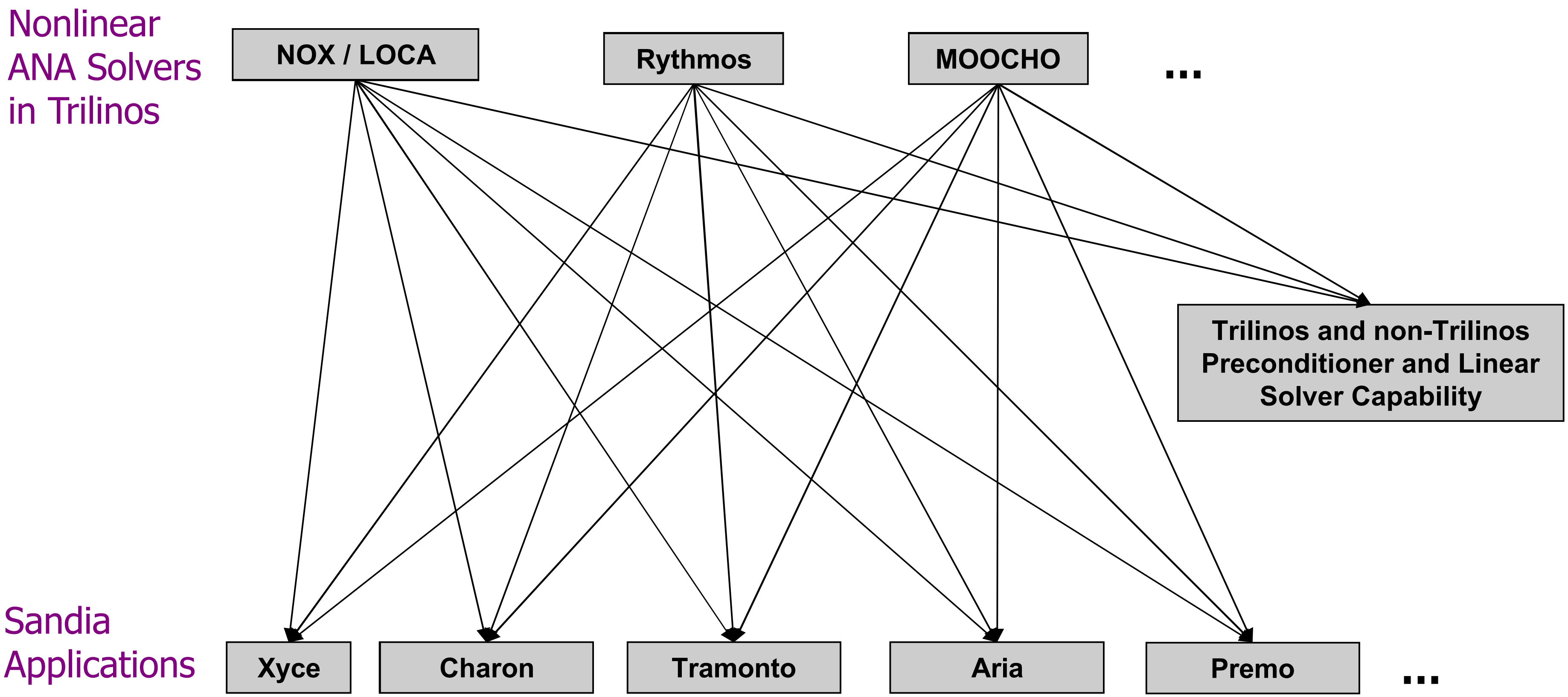
$$W = \beta \frac{\partial f}{\partial x} = \begin{bmatrix} \beta \frac{\partial \tilde{f}^1}{\partial \tilde{x}^1} & \\ \beta \frac{\partial \tilde{f}^2}{\partial \tilde{p}_1^2} & \beta \frac{\partial \tilde{f}^2}{\partial \tilde{x}^2} \end{bmatrix}$$

Nonsingular linear operators on the diagonal

## "Composite" ANA Subclasses:



# Nonlinear Algorithms and Applications : Everyone for Themselves?

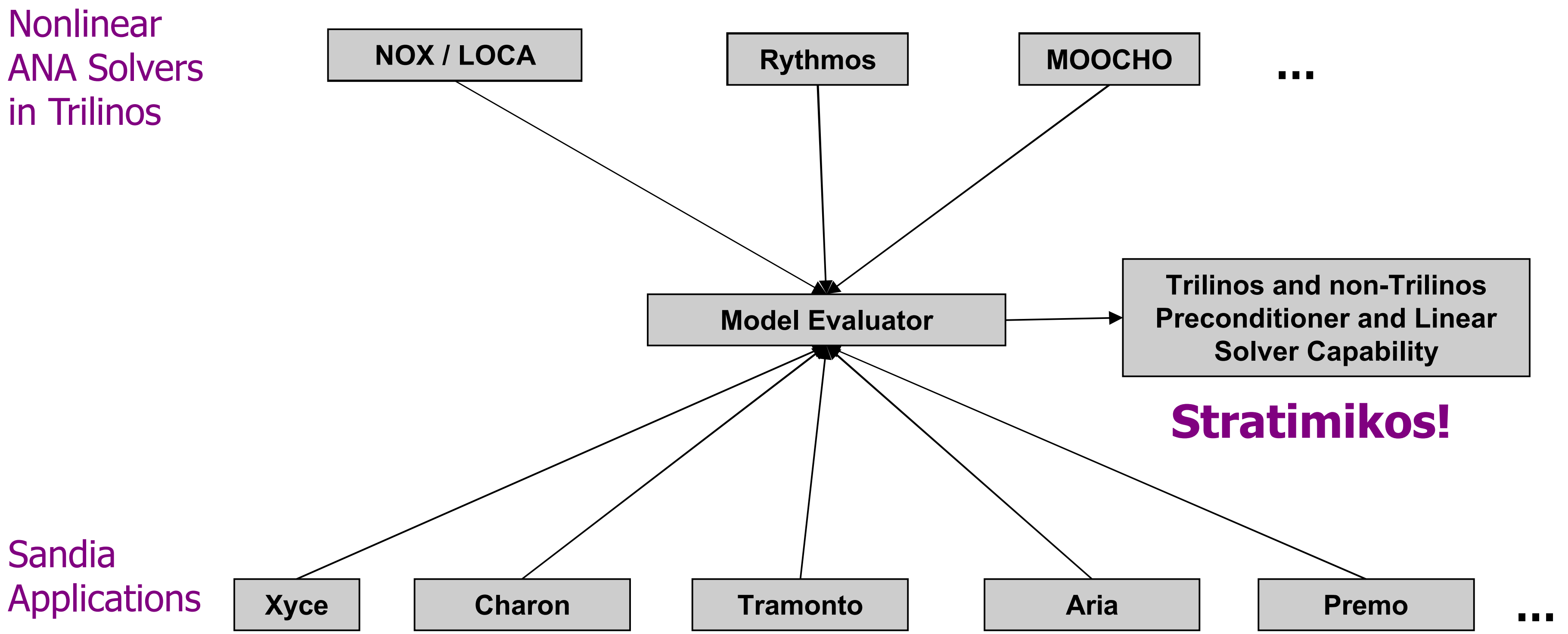



**Key Point**

- BAD

Sandia National Laboratories

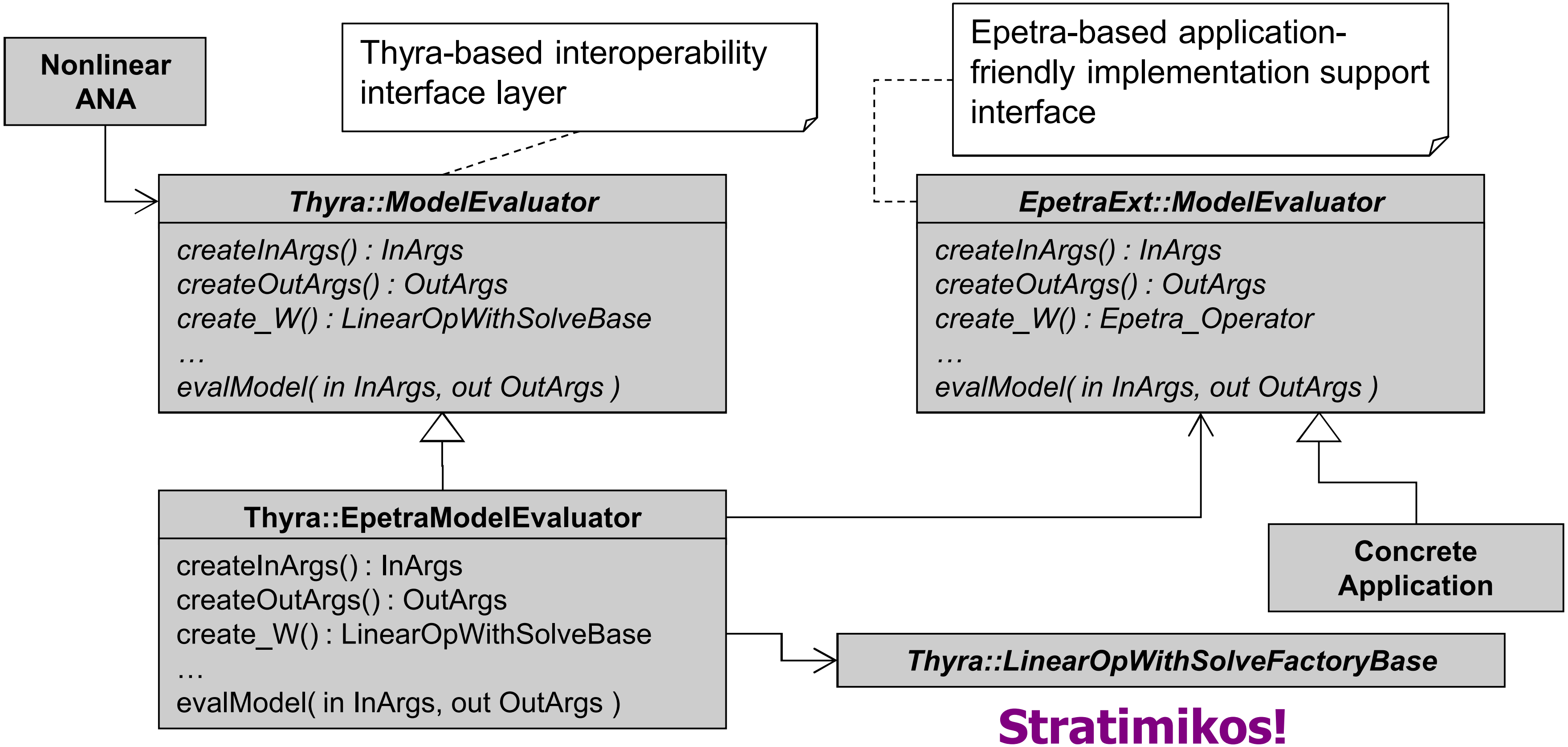
# Nonlinear Algorithms and Applications : Thyra & Model Evaluator!




**Key Points**

- Provide single interface from nonlinear ANAs to applications
- Provide single interface for applications to implement to access nonlinear ANAs
- Provides shared, uniform access to linear solver capabilities
- Once an application implements support for one ANA, support for other ANAs can quickly follow

# Model Evaluator : Thyra and EpetraExt Versions




- Thyra::ModelEvaluator and EpetraExt::ModelEvaluator are near mirror copies of each other.
- Thyra::EpetraModelEvaluator is fully general adapter class that can use any linear solver through a Thyra::LinearOpWithSolveFactoryBase object it is configured with
- Stateless model that allows for efficient multiple shared calculations (e.g. automatic differentation)
- Adding input and output parameters involves
  - Modifying only the classes Thyra::ModelEvaluator, EpetraExt::ModelEvaluator, and Thyra::EpetraModelEvaluator
  - Only recompilation of Nonlinear ANA and Concrete Application code

Sandia National Laboratories

# Example EpetraExt::ModelEvaluator Application Implementation

```
/** \brief Simple example ModelEvaluator subclass for a 2x2 set of
 * parameterized nonlinear equations.
 *
 * The equations modeled are:
 \verbatim

    f[0] =       x[0]      + x[1]*x[1] - p[0];
    f[1] = d * ( x[0]*x[0] - x[1]      - p[1] );

 \endverbatim
 */
class EpetraModelEval2DSim : public EpetraExt::ModelEvaluator {
public:
  EpetraModelEval2DSim(...);
  /** \name Overridden from EpetraExt::ModelEvaluator . */
  //@{
  Teuchos::RefCountPtr<const Epetra_Map>      get_x_map() const;
  Teuchos::RefCountPtr<const Epetra_Map>      get_f_map() const;
  Teuchos::RefCountPtr<const Epetra_Vector>   get_x_init() const;
  Teuchos::RefCountPtr<Epetra_Operator>       create_W() const;
  InArgs      createInArgs() const;
  OutArgs     createOutArgs() const;
  void evalModel( const InArgs& inArgs, const OutArgs& outArgs ) const;
  //@}

private:
  ...
};
```

Complete nonlinear equations example in epetraext/thyra/example/model_evaluator/2dsim/.

Sandia National Laboratories

# Example EpetraExt::ModelEvaluator Application Implementation

```
EpetraExt::ModelEvaluator::InArgs EpetraModelEval2DSim::createInArgs() const
{
  InArgsSetup inArgs;
  inArgs.setModelEvalDescription(this->description());
  inArgs.setSupports(IN_ARG_x,true);
  inArgs.setSupports(IN_ARG_beta,true);
  return inArgs;
}

EpetraExt::ModelEvaluator::OutArgs EpetraModelEval2DSim::createOutArgs() const
{
  OutArgsSetup outArgs;
  outArgs.setModelEvalDescription(this->description());
  outArgs.setSupports(OUT_ARG_f,true);
  outArgs.setSupports(OUT_ARG_W,true);
  outArgs.set_W_properties(
    DerivativeProperties(DERIV_LINEARITY_NONCONST,DERIV_RANK_FULL,true)
    );
  return outArgs;
}

void EpetraModelEval2DSim::evalModel( const InArgs& inArgs, const OutArgs& outArgs ) const
{
  const Epetra_Vector    &x     = *inArgs.get_x();
  Epetra_Vector          f_out = outArgs.get_f().get();
  Epetra_Operator        W_out = outArgs.get_W().get();
  if(f_out) {
    ...
  }
  if(W_out) {
    ...
  }
}
```

**Key Point**

From looking at example code, there is not even a hint that other input and output parameters exist!

# Model Derivatives

## First Derivatives

- State function state sensitivities:

$$W = \alpha \frac{\partial f}{\partial \dot{x}} + \beta \frac{\partial f}{\partial x}$$

[LinearOpWithSolveBase or LinearOpBase]

- State function parameter sensitivities:

$$\frac{\partial f}{\partial p_l}, \text{ for } l = 1 \ldots N_p$$

[LinearOpBase or MultiVectorBase]

- Auxiliary function state sensitivities:

$$\frac{\partial g_j}{\partial x}, \text{ for } j = 1 \ldots N_g$$

[LinearOpBase or MultiVectorBase$^2$]

- Auxiliary function parameter sensitivities:

$$\frac{\partial g_j}{\partial p_l}, \text{ for } j = 1 \ldots N_g,\ l = 1 \ldots N_p$$

[LinearOpBase or MultiVectorBase$^2$]

## Use Cases:

- Steady-state and transient sensitivity computations
- Optimization
- Multi-physics coupling
- …

# Forward/Direct and Adjoint Sensitivities

Steady-state constrained response:

$$g(x,p) \text{ s.t. } f(x,p) = 0$$

Reduced response function:

$$\hat{g}(p) = g(x(p), p)$$

Reduced Sensitivities:

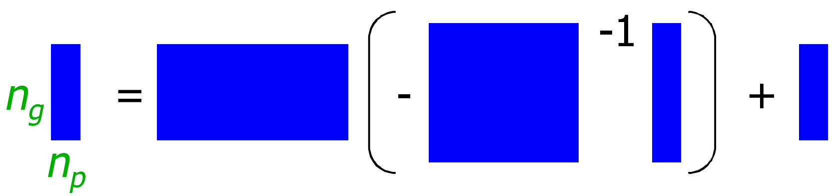
$$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial g}{\partial p} \quad \text{where:} \quad \frac{\partial x}{\partial p} = -\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}$$

**Key Point**

The form of the derivatives you need depends on whether you are doing direct or adjoint sensitivities

## Forward/Direct Sensititivies ($n_g$ large, $n_p$ small)

$$\frac{\partial \hat{g}}{\partial p} = \frac{\partial g}{\partial x}\left(-\frac{\partial f}{\partial x}^{-1}\frac{\partial f}{\partial p}\right) + \frac{\partial g}{\partial p}$$

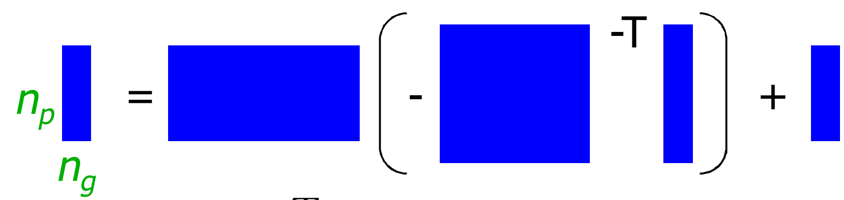



| $\frac{\partial \hat{g}}{\partial p}$ | $\frac{\partial g}{\partial x}$ | $\frac{\partial f}{\partial x}$ | $\frac{\partial f}{\partial p}$ | $\frac{\partial g}{\partial p}$ |
|---|---|---|---|---|
| MV | LO | LOWS | MV | MV |

(LOWS, MV) → MV

## Adjoint Sensititivies ($n_g$ small, $n_p$ large)

$$\frac{\partial \hat{g}}{\partial p}^T = \frac{\partial f}{\partial p}^T\left(-\frac{\partial f}{\partial x}^{-T}\frac{\partial g}{\partial x}^T\right) + \frac{\partial g}{\partial p}^T$$




| $\frac{\partial \hat{g}}{\partial p}^T$ | $\frac{\partial f}{\partial p}^T$ | $\frac{\partial f}{\partial x}$ | $\frac{\partial g}{\partial x}^T$ | $\frac{\partial g}{\partial p}^T$ |
|---|---|---|---|---|
| MV | LO | LOWS | MV | MV |

(LOWS, MV) → MV

Sandia National Laboratories

# Properties of Current Approach to ModelEvaluator Software

- Strong Typing of Input/Ouput Object Types but Weak Typing of Problem Formulation
  - Much functionality/information resides in concrete InArgs and OutArgs classes
  - ModelEvaluator objects select which input/output arguments are recognized and the rest are ignored
  - Attempts to set or get non-supported input/output arguments throw exceptions as early as possible and result in very good error messages
  - Only subclasses of ModelEvaluator can change the set of supported arguments
- Designed for Change
  - Input and output arguments can be added at will to support new algorithms; only requires recompilation of existing clients and subclasses
- Incremental Development of Application Capabilities
  - Existing ModelEvaluator subclasses can incrementally take on new input and output objects to support more advanced algorithm capabilities

    => Gradual addition of new function overrides and expansion of the implementations of createInArgs(), createOutArgs(), and evalModel(...).
- Self-Describing Models => Smart Algorithms
  - Clients can query InArgs and OutArgs objects to see what input and output objects are supported
  - Properties of derivative objects is provided by OutArgs object!
- Independence/Multiplicity of Input and Output Objects
  - Input and output objects are independent of the ModelEvaluator object and as many or as few as required are created on demand by the client

Sandia National Laboratories

# Impact of the Nonlinear Model Evaluator

- Incorporation into simulation codes
  - Charon: QASPR project (Hoekstra(1437),…) => ASC Level-2 Milestone
  - Rapid Production CSRF (Bartlett(1411), vBW(1411), Long(8962), Phipps(1416), …)
  - Aria/SIERRA (Notz(1514), Hooper(1416))
  - Tramonto: Decontamination LDRD (vBW(1411),…)
  - …
- Incorporation into numerical algorithms
  - MOOCHO: Simulation-constrained optimization (Bartlett(1411))
  - Rythmos: Time integration and sensitivity methods (Coffey(1414))
  - NOX: Nonlinear equation solvers (Pawlowski(1416))
  - LOCA: Library of continuation algorithms (Salinger(1416), Phipps(1416))
  - Aristos: Full-space simulation-constrained optimization (Ridzal(1414))
  - …
- Connection with other SNL projects
  - 4D CSRF, Transient to steady-state (Salinger(1416), Dunlavy(1411))
  - Multi-physics LDRD (Hooper(1416), Pawlowski(1416))
  - …

Sandia National Laboratories

# Outline

- Mathematical overview of sensitivities and optimization
- Minimally invasive optimization algorithm for MOOCHO
- ModelEvaluator software
- Wrap it up

# ModelEvaluator Software Summary

- Motivation for Unified ModelEvaluator Approach to Nonlinear Problems
  - Large overlap in commonality between requirements for many nonlinear abstract numerical algorithms (ANAs).
  - Mixed problem types will become more and more common and must be easy to support

- Properties of ModelEvalutor Software
  - Strong Typing of Input/Object Types but Weak Typing of Problem Formulation
  - Designed for Change
  - Incremental Development of Application Capabilities
  - Self-Describing Models => Smart Algorithms
  - Independence/Multiplicity of Input and Output Objects

- ANAs already using or can use ModelEvaluator
  - MOOCHO (constrained optimization, unconstrained optimization, nonlinear equations)
  - Rythmos (explicit ODEs, implicit ODEs, DAEs)
  - NOX (nonlinear equations)
  - LOCA (stability analysis, continuation)
  - Aristos (full space, trust-region optimization)

# Sensitivity and Optimization Summary

- Need to go beyond the forward solve to answer:
  - How to a characterize the error in my model so that it can be improved? → Error estimation
  - What is the uncertainty in *x* given uncertainty in *p* ? → QMU
  - What is the "best" value of *p* so that my model *f(x,p)=0* fits exp. data? → Param. Estimation
  - What is the "best" value for *p* to achieve some goal? → Optimization
- Sensitivities
  - Direct vs. Adjoint methods
- Optimization methods
  - Decoupled (DAKOTA) vs. Coupled (MOOCHO, Aristos)
  - Coupled methods: Full-space (Aristos) vs. reduced-space (MOOCHO)
  - Inequality constraints (MOOCHO active-set methods)
- Minimally invasive optimization method for MOOCHO
  - Requires only forward state Jacobian solves and objective evaluations
  - All other computations can be approximated with directional finite differences