SAND2014-19572 C

Rapid Optimization Library Rapid Optimization Library Rapid Optimization Library Rapid Optimization Library

Rapid Optimization Library Rapid Optimization Library Rapid Optimization Library

Rapid Optimization Library

Drew Kouri, Denis Ridzal
Bart van Bloemen Waanders, Greg von Winckel

Sandia National Laboratories
Optimization & UQ, Org. 1441

Version: Trilinos 12.2, July 2015

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2014-XXXXP

- **What is ROL?**
- **Motivation**
- **Problem formulations**
- **Application programming interface**
- **Methods**
- **Research focus**

Rapid Optimization Library (ROL)

- ROL is a Trilinos package for large-scale continuous optimization, a.k.a. nonlinear programming (NLP).
- Available in Trilinos since 10/21/2014.
- ROL includes:
  - A rewrite and consolidation of existing optimization tools in Trilinos: *Aristos*, *MOOCHO*, *Optipack*, *Globipack.*
  - Hardened, production-ready algorithms for unconstrained and equality-constrained continuous optimization.
  - Methods for efficient handling of inequality constraints.
  - A unified interface for simulation-based optimization.
  - New methods for efficient handling of inexact computations.
  - New methods for optimization under uncertainty.

Motivation

- Optimization of differentiable simulated processes:
  - partial differential equations (PDEs)
  - differential algebraic equations (DAEs)
  - network equations (gas pipelines, electrical networks)
- Inverse problems, model calibration.
- Optimal design, including topology and shape optimization.
- Optimal control, optimal design of experiments, etc.
- Parameter/design/control spaces can be very large, often related to the size of the computational mesh (PDEs) or the size of the device network or graph (DAEs).
- Simulated processes may be subject to uncertainty.

Motivation

- Example cost of deterministic optimization, in terms of "simulation units", such as nonlinear PDE/DAE solves:

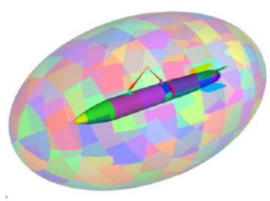
Information	Size of parameter space: 1	10	10^3	10^6	Methods
Function samples (incl. finite diff's)	100	10,000	∞	∞	Global search or steepest descent
Analytic gradients (hand-coded or AD)	50	100	200	1,000	Quasi-Newton
Analytic Hessians (hand-coded or AD)	50	50	50	50	Newton-Krylov

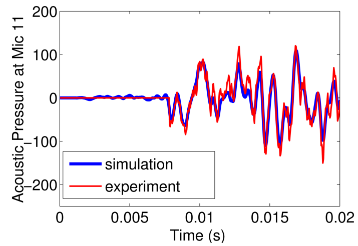
- *We want derivative-based methods.*
- *We want **embedded and matrix-free** methods:*
  - Direct access to application data structures: vectors, etc.
  - Direct use of application methods: (non)linear solvers, etc.

A few current use cases

Inverse problems in acoustics / elasticity.

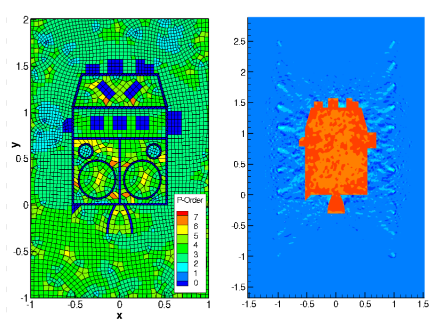
- Interface to the Sierra-SD structural dynamics code (Sandia, Org. 1500).





1M optimization variables, 1M state variables

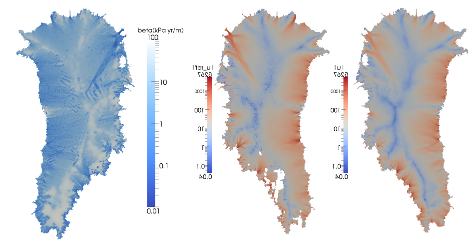
- Interface to DGM, a high-order DG code (Sandia, Org. 1400).



- **175K distributed optimization variables**
- 525K x 10K state variables

Estimating basal friction of ice sheets.

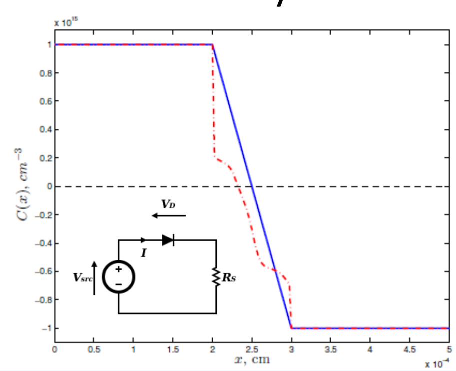
- Interface to LifeV Project (www.lifev.org).



- **65K distributed optimization variables**
- 700K state variables

Calibration of electrical device models.

- Interface to Xyce circuit simulator.



- **50 optimization variables (in-memory or disk storage)**

Mathematical abstraction

- Straight from ROL's documentation:

ROL is used for the numerical solution of smooth optimization problems

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & c(x) = 0\,, \\ & a \le x \le b\,, \end{aligned}$$

where:

- $f : \mathcal{X} \to \mathbb{R}$ is a Fréchet differentiable functional,
- $c : \mathcal{X} \to \mathcal{C}$ is a Fréchet differentiable operator,
- $\mathcal{X}$ and $\mathcal{C}$ are Banach spaces of functions, and
- $a \le x \le b$ defines pointwise (componentwise) bounds on x.

- This abstraction is a valuable guiding principle.

Problem formulations

- ROL supports four basic NLP problem types:

Type-U: Unconstrained.

$$\min_x \quad f(x)$$

Type-B: Bound constrained.

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & a \le x \le b \end{aligned}$$

Type-E: Equality constrained.

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & c(x) = 0 \end{aligned}$$

Type-EB: Equalities + bounds.

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & c(x) = 0 \\ & a \le x \le b \end{aligned}$$

Note:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & c(x) \le 0 \end{aligned} \qquad \longleftrightarrow \qquad \begin{aligned} \min_{x,s} \quad & f(x) \\ \text{subject to} \quad & c(x) + s = 0\,, \\ & s \ge 0\,. \end{aligned}$$

Design of ROL

Application programming interface

Linear algebra interface	Functional interface		Algorithmic interface
Vector	Objective BoundConstraint EqualityConstraint	SimOpt Middleware	StatusTest **Step** DefaultAlgorithm

Methods – Implementations of Step instances

Linear algebra interface

- ROL::Vector is designed to enable direct use of application data structures (serial, parallel, in-memory, disk-based, etc.).
- Methods:
  plus, scale, dot, norm, clone (pure virtual)
  axpy, zero, set (virtual)
  basis, dimension (optional)
- Nothing new. History: HCL/RVL, TSFCore, Thyra.
- Recent applications of ROL require dual-space operations:
  dual **(virtual)**
- Note: Other Trilinos packages have similar linear algebra interfaces, but may not be able to take advantage of dual-space operations, such as Riesz maps.

Functional interface

$$\begin{aligned} \min_{x} \quad & \boxed{f(x)} \\ \text{subject to} \quad & c(x) = 0 \\ & a \le x \le b \end{aligned}$$

- ROL::Objective provides the objective function interface.
- Methods:
  value (pure virtual)
  gradient, hessVec (virtual)
  update, invHessVec, precond, dirDeriv (optional)
- We can use finite differences to approximate missing derivative information (default implementation).
- For best performance, implement analytic derivatives.
- Tools: checkGradient, checkHessVec, checkHessSym.
- ROL::BoundConstraint enables pointwise bounds on optimization variables, in support of projected gradient, projected Newton, and primal-dual active set methods.

Functional interface

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & \boxed{c(x) = 0} \\ & a \leq x \leq b \end{aligned}$$

- ROL::EqualityConstraint enables equality constraints.
- Methods:
  value (pure virtual)
  applyJacobian, applyAdjointJacobian, applyAdjointHessian (virtual)
  update, applyPreconditioner, solveAugmentedSystem (optional)
- We can use finite differences to approximate missing derivative information (default implementation).
- For best performance, implement analytic derivatives.
- Tools: checkApplyJacobian, etc.

Functional interface

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & \boxed{c(x) = 0} \\ & a \le x \le b \end{aligned}$$

- Documentation excerpt:

template<class Real >

```
void ROL::EqualityConstraint< Real >::applyAdjointHessian ( Vector< Real > &        ahuv,
                                                            const Vector< Real > &  u,
                                                            const Vector< Real > &  v,
                                                            const Vector< Real > &  x,
                                                            Real &                  tol
                                                          )
```

virtual

Apply the derivative of the adjoint of the constraint Jacobian at x to vector u in direction v, according to $\boxed{v \mapsto c''(x)(v,\cdot)^* u.}$

Parameters

[out]	**ahuv**	is the result of applying the derivative of the adjoint of the constraint Jacobian at **x** to vector **u** in direction **v**; a dual optimization-space vector
[in]	**u**	is the direction vector; a dual constraint-space vector
[in]	**v**	is an optimization-space vector
[in]	**x**	is the constraint argument; an optimization-space vector
[in,out]	**tol**	is a tolerance for inexact evaluations; currently unused

On return, ahuv $= c''(x)(v,\cdot)^* u$, where $u \in \mathcal{C}^*$, $v \in \mathcal{X}$, and ahuv $\in \mathcal{X}^*$.

The default implementation is a finite-difference approximation based on the adjoint Jacobian.

SimOpt: The middleware for engineering optimization

- Many simulation-based Type-E problems have the form:

$$\min_{u,z} \; f(u,z) \quad \text{subject to} \quad c(u,z) = 0$$

  - u denote simulation variables (state variables, **basic, Sim**)
  - z denote optimization variables (controls, parameters, **nonbasic, Opt**)

- A common Type-U reformulation, by nonlinear elimination, is:

$$\min_{z} \; f(u(z),z) \quad \text{where } u(z) \text{ solves } c(u,z) = 0$$

- For these cases, the SimOpt interface enables direct use of methods for **both** unconstrained and constrained problems.

SimOpt: The middleware for engineering optimization

Objective_SimOpt

value(u,z)
gradient_1(g,u,z)
gradient_2(g,u,z)
hessVec_11(hv,v,u,z)
hessVec_12(hv,v,u,z)
hessVec_21(hv,v,u,z)
hessVec_22(hv,v,u,z)

Note: 1 = Sim = u
2 = Opt = z

EqualityConstraint_SimOpt

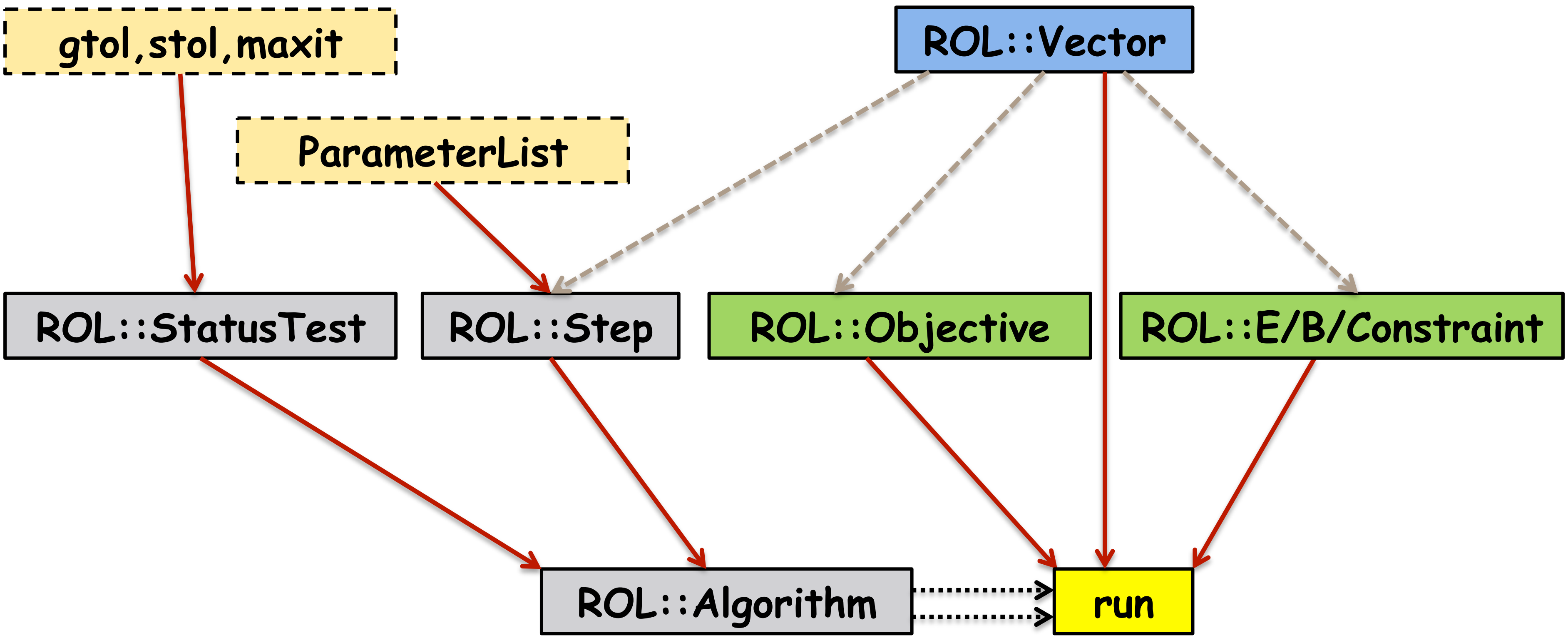
value(c,u,z)
applyJacobian_1(jv,v,u,z)
applyJacobian_2(jv,v,u,z)
applyInverseJacobian_1(ijv,v,u,z)
applyAdjointJacobian_1(ajv,v,u,z)
applyAdjointJacobian_2(ajv,v,u,z)
applyInverseAdjointJacobian_1(iajv,v,u,z)
applyAdjointHessian_11(ahwv,w,v,u,z)
applyAdjointHessian_12(ahwv,w,v,u,z)
applyAdjointHessian_21(ahwv,w,v,u,z)
applyAdjointHessian_22(ahwv,w,v,u,z)
solve(u,z)

SimOpt: Benefits

- Streamlined modular implementation for a very large class of engineering optimization problems.
- Implementation verification through a variety of ROL tests:
  - Finite difference checks with high granularity.
  - Consistency checks for operator inverses and adjoints.
- Access to all optimization methods through a **single interface**.
- Enables future ROL interfaces for advanced solution checkpointing and restarting, closer integration with application-specific time integrators, etc.

Algorithmic interface

- Modular design:



Algorithmic interface

- An illustration, sans details, using a sequential quadratic programming (SQP) step for Type-E formulations:

```
RCP<Objective<RealT> > obj;
RCP<EqualityConstraint<RealT> > constr;

RCP<CompositeStepSQP<RealT> > step(parlist);
RCP<StatusTestSQP<RealT> > status(gtol, ctol, stol, maxit);

DefaultAlgorithm<RealT> algo(step, status);

x.zero(); vl.zero();

algo.run(x, vl, *obj, *constr);
```

Methods – Part 1

- **Type-U (unconstrained):**
  - Globalization: LineSearchStep and TrustRegionStep.
  - Gradient descent, quasi-Newton (limited-memory BFGS, DFP, Barzilai-Borwein), nonlinear CG (6 variants), inexact Newton (including finite difference hessVecs), Newton, with line searches and trust regions.
  - Trust-region methods supporting inexact objective functions and inexact gradient evaluations. Enables *adaptive and reduced models*.

- **Type-B (bound constrained):**
  - Projected gradient and projected Newton methods.
  - Primal-dual active set methods.

Methods – Part 2

- **Type-E (equality constrained):**
  - Sequential quadratic programming (SQP) with trust regions, supporting inexact linear system solves.
  - A hierarchy of full-space SQP methods, based on the constraint null-space representation (summer 2015):
    (1) sim/opt splitting with simple linearized forward and adjoint solves,
    (2) simple optimality systems with forward/adjoint preconditioners,
    (3) full KKT (optimality) system solves.

- **Type-EB (equality + bound constrained):**
  - Augmented Lagrangian methods.
  - Semismooth Newton methods (summer 2015).
  - Interior-point methods (summer 2015).

Methods – Part 3

- **Optimization under uncertainty:**

$$\min_{z} \; \sigma(f(z,\vartheta))$$

$$\min_{z} \; \sigma(f(u(z,\vartheta),z,\vartheta)) \quad \text{where } u(z,\vartheta) \text{ solves } c(u,z,\vartheta)=0$$

  - Compute controls/designs that are risk-averse or robust to uncertainty in the parameters ϑ. Here σ is some **risk measure**.
  - Risk measures: Conditional value-at-risk (CVaR), Expectation (mean), Mean plus deviation, Mean plus variance, Exponential disutility.
  - Incorporate sampling and adaptive quadrature approaches from uncertainty quantification. Flexible sampling interface through `SampleGenerator` and `BatchManager`.
  - Control inexactness and adaptivity through **trust-region** framework.

Research focus

- Optimization under uncertainty, risk-averse optimization.
- Treatment of general constraints in large-scale optimization.
- Sequential subspace methods, continuation, regularization.
- Inexact and adaptive methods for large-scale optimization.
- Tighter application integration through SimOpt.

Miscellaneous

- Efficient computations, restarts and checkpointing enabled through `AlgorithmState` and `StepState`.
- Flexible output using user-defined streams.
- Soft and hard iteration updates are possible, for efficiency.

- Coming in 2015:
  - Specialized techniques for topology optimization, such as generalizations of method of moving asymptotes (MMA).
  - Computing conservative estimates of probability of failure, through buffered probabilities.
  - Methods for general constraints.
  - Hierarchy of full-space SQP methods.
  - User's guide.