MOOC CHO
and
Object-Oriented Interfaces and Algorithms for SAND PDE-Constrained Nonlinear Programming

Roscoe A. Bartlett
Senior Member Technical Staff
Optimization and Uncertainty Estimation Department
Sandia National Laboratories
Outline

- Overview of MOOCHO
- Requirements for levels of optimization => Linear algebra Interfaces
- Vector reduction/transformation operators (RTOp)
- Timing experiments
Outline

- Overview of MOOCHO
- Requirements for levels of optimization => Linear algebra Interfaces
- Vector reduction/transformation operators (RTOp)
- Timing experiments
Nonlinear Programming for Large-Scale Optimization

Nonlinear program (NLP) formulation:

\[
\begin{align*}
\min & \quad f(x) \\
\text{s.t.} & \quad c(x) = 0 \\
& \quad x_L \leq x \leq x_U
\end{align*}
\]

\[ f(x) : \mathbb{R}^n \rightarrow \mathbb{R} \]
\[ c(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

• Classes of problems considered:
  • Large-scale (up to \(10^6\) variables and more)
  • Special problem structure (e.g. PDEs, DAEs)
  • Application specific methods (e.g. linear solvers, nonlinear globalization)
  • Specialized computing environments (e.g. MPP, client/server, out-of-core)

• What we can not do:
  • Use the current generation of optimization software to exploit problem structure and take advantage of advanced computing environments

• What we do not want to do:
  • Write a new optimization implementation for each piece of application software

• What we do want to do:
  • Use gradient-based methods for simultaneous analysis and design (SAND)
Desirable features:

- Provide sophisticated well-tested methods (NLP independent)
  - Globalization methods (i.e. line searches and/or trust regions)
  - Step modifications (i.e. negative curvature)
- Allow users to specialize parts of the algorithm that are NLP specific
  - Linear solvers (direct or iterative, serial or parallel)
  - Data structures (serial or parallel) => Fully scalable algorithms
Introducing MOOCHO!

**MOOCHO**: Multifunctional Object-Oriented arCHitecture for Optimization

- Built using object-oriented principles => OO!
- Initially developed at CMU (rSQP++) => Good times!
- Active-set and interior-point SQP-related methods => Fast! or Fast?
- Open source (spring/summer 2003) => You can get it!
- Flexible algorithm configuration => You can alter it!
- Exchangeable numerical components => You can tailor it!
- Abstract linear-algebra interfaces => You can parallelize it!
Outline

- Overview of MOOCHO
- Requirements for levels of optimization => Linear algebra Interfaces
- Vector reduction/transformation operators (RTOp)
- Timing experiments
Requirements for Simultaneous Analysis and Design (SAND)

Standard NLP formulation

\[
\begin{align*}
\text{min} & \quad f(x) \\
\text{s.t.} & \quad c(x) = 0 \\
& \quad x_L \leq x \leq x_U
\end{align*}
\]

\[f(x) : \mathcal{X} \rightarrow \mathbb{R}\]
\[c(x) : \mathcal{X} \rightarrow \mathbb{C}\]
\[\mathcal{X} \subseteq \mathbb{R}^n, \quad \mathcal{C} \subseteq \mathbb{R}^m\]

NLP formulation for PDE-Constrained Optimization

\[
\begin{align*}
\text{min} & \quad f(y,u) \\
\text{s.t.} & \quad c(y,u) = 0 \\
& \quad (y_L, u_L) \leq (y, u) \leq (y_U, u_U)
\end{align*}
\]

\[f(y,u) : \mathcal{Y} \times \mathcal{U} \rightarrow \mathbb{R}\]
\[c(y,u) : \mathcal{Y} \times \mathcal{U} \rightarrow \mathbb{C}\]
\[\mathcal{Y} \subseteq \mathbb{R}^m, \quad \mathcal{U} \subseteq \mathbb{R}^{n-m}, \quad \mathcal{C} \subseteq \mathbb{R}^m\]

where: \(\mathcal{X} = \mathcal{Y} \times \mathcal{U}\)

Definitions

- Basis \(C\) and nonbasis \(N\) matrices:
  \[\nabla c^T = [C, N]\s.t. C\text{ is nonsingular}\]

- Hessian of the Lagrangian \(W\):
  \[W = \nabla^2 f(x) + \Sigma \lambda_j \nabla^2 c_j(x)\]

Requirements (Levels of Opt. Methods)

**Adjoint SAND**

- \(q = C p\) and \(p = C^T q\), where: \(p \in \mathcal{Y}, q \in \mathcal{C}\)
- \(q = N p\) and \(p = N^T q\), where: \(p \in \mathcal{U}, q \in \mathcal{C}\)
- \(q = C^{-1} p\) and \(q = C^{-T} p\), where: \(p \in \mathcal{C}, q \in \mathcal{Y}\)

**Full-Newton SAND**

- \(q = W_{yy} p\), where: \(p \in \mathcal{Y}, q \in \mathcal{Y}\)
- \(q = W_{yu} p\) and \(p = W_{yu}^T q\), where: \(p \in \mathcal{U}, q \in \mathcal{Y}\)
- \(q = W_{uu} p\), where: \(p \in \mathcal{U}, q \in \mathcal{U}\)

**Vectors!!!**
MOOCHO NLP (Application) Interfaces

**Question: How are following represented?**

- **Vectors spaces**: $\mathcal{X}, \mathcal{C}$
- **Vectors**: $x, x_L, x_U, c, \nabla f, p$
- **Matrices**: $\nabla c, N, D, W$
- **Invertible matrices**: $C$

**AbstractLinAlgPack::VectorSpace**
- **space_x**
- **space_c**

**AbstractLinAlgPack::BasisSystem**
- **basis_sys**

**MOOCHO NLP (Application) Interfaces**

**min** $f(x)$
**s.t.** $c(x) = 0$
$x_L < x < x_U$

* Initial guess $x_0$
* Variable bounds $x_L, x_U$
* Function evaluations $f(x), c(x)$

* Objective gradient $\nabla f(x)$

* General constraints gradients matrix $\nabla c(x)$

* Hessian of the Lagrangian matrix
$W = \nabla^2 f(x) + \Sigma \lambda_j \nabla^2 c_j(x)$

**NLP**
- $x_{\text{init}} : \text{Vector}$
- $x_l : \text{Vector}$
- $x_u : \text{Vector}$
- $\text{calc}_f(in \ x)$
- $\text{calc}_c(in \ x)$

**NLPObjGrad**
- $\text{calc}_Gf(in \ x)$

**NLPFirstOrder**
- $\text{calc}_Gc(in \ x)$

**NLPSecondOrder**
- $\text{calc}_HL(in \ x, \text{in lambda})$

**NLPDirect**
- $\text{calc}_\text{point}(in \ x, \text{out f, out c, out Gf, out py, out D})$

**Unified Modeling Language (UML) Class Diagram**
(Nearly) Compatible Interface (C++) Implementations

- HCL (Hilbert Class Library)
- TSF (Trilinos Solver Framework)
- AbstractLinAlgPack => MOOCHO linear algebra interface
Outline

- Overview of MOOCHO
- Requirements for levels of optimization => Linear algebra Interfaces
- Vector reduction/transformation operators (RTOp)
- Timing experiments
Examples of Non-Standard Vector Operations

Examples from OOQP (Gertz, Wright)

\[
y_i \leftarrow y_i + \alpha x_i z_i, \quad i = 1...n
\]
\[
y_i \leftarrow \begin{cases} y_{\min} - y_i & \text{if } y_i < y_{\min} \\ y_{\max} - y_i & \text{if } y_i > y_{\max} \\ 0 & \text{if } y_{\min} \leq y_i \leq y_{\max} \end{cases}, \quad i = 1...n
\]
\[
y_i \leftarrow y_i / x_i, \quad i = 1...n
\]
\[
\alpha \leftarrow \{ \max \alpha : x + \alpha d \geq \beta \}
\]

Example from TRICE (Dennis, Heinkenschloss, Vicente)

\[
d_i \leftarrow \begin{cases} (b - u)_{i}^{1/2} & \text{if } w_i < 0 \text{ and } b_i < +\infty \\ 1 & \text{if } w_i < 0 \text{ and } b_i = +\infty \\ (u - a)_{i}^{1/2} & \text{if } w_i \geq 0 \text{ and } a_i > -\infty \\ 1 & \text{if } w_i \geq 0 \text{ and } a_i = -\infty \end{cases}, \quad i = 1...n
\]

Example from IPOPT (Waechter)

\[
x_i \leftarrow \begin{cases} \left( x_i^L + \frac{(x_i^U - x_i^L)}{2} \right) & \text{if } \hat{x}_i^L \geq \hat{x}_i^U \\ \hat{x}_i^L & \text{if } x_i < \hat{x}_i^L \\ \hat{x}_i^U & \text{if } x_i > \hat{x}_i^U \end{cases}, \quad i = 1...n
\]

Currently in MOOCHO:

> 40 vector operations!
<table>
<thead>
<tr>
<th>Goal</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute efficiency</td>
<td>Near optimal performance</td>
</tr>
<tr>
<td>Optimization developers add new operations</td>
<td>Independence of linear algebra library developers</td>
</tr>
<tr>
<td>Compute environment independence</td>
<td>Flexible optimization software</td>
</tr>
<tr>
<td>Minimal number of methods</td>
<td>Easy to write adapters</td>
</tr>
</tbody>
</table>
(1) Linear algebra library allows direct access to vector elements

(2) Optimizer-specific interfaces

(3) General-purpose primitive vector operations
Vector Reduction/Transformation Operators Defined

Reduction/Transformation Operators (RTOp) Defined

\[ z^{1\ldots i} \ldots z^{q\ldots i} \leftarrow \text{op}_t(i, v^{1\ldots i} \ldots v^{p\ldots i}, z^{1\ldots i} \ldots z^{q\ldots i}) \]  
\[ \beta \leftarrow \text{op}_r(i, v^{1\ldots i} \ldots v^{p\ldots i}, z^{1\ldots i} \ldots z^{q\ldots i}) \]  
\[ \beta^2 \leftarrow \text{op}_r(\beta^1, \beta^2) \]

- element-wise transformation
- element-wise reduction
- reduction of intermediate reduction objects

\[ \bullet \ v^{1\ldots p} \in R^n : p \text{ non-mutable input vectors} \]
\[ \bullet \ z^{1\ldots q} \in R^n : q \text{ mutable input/output vectors} \]
\[ \bullet \ \beta : \text{ reduction target object (many be non-scalar (e.g. } \{y_k,k\}), \text{ or NULL)} \]

Key to Optimal Performance

- \( \text{op}_t(\ldots) \) and \( \text{op}_r(\ldots) \) applied to entire sets of subvectors \((i = a\ldots b)\) independently:
  \[ z^{1\ldots a:b} \ldots z^{q\ldots a:b}, \beta \leftarrow \text{op}(a, b, v^{1\ldots a:b} \ldots v^{p\ldots a:b}, z^{1\ldots a:b} \ldots z^{q\ldots a:b}, \beta) \]
- Communication between sets of subvectors only for \( \beta \neq \text{NULL} \), \( \text{op}_r(\beta^1, \beta^2) \rightarrow \beta^2 \)
Object-Oriented Design for User Defined RTOp Operators

**Advantages:**
- **Functionality**
  - Linear-algebra implementations can be changed with no impact on optimizer
  - Optimizer developers can unilaterally add new vector operations
- **Performance**
  - Near optimal performance (large subvectors)
  - Multiple simultaneous global reductions => no sequential bottlenecks
  - No unnecessary temporary vectors or multiple vector read/writes
- **Disadvantages:**
  - New concepts, initially harder to understand interfaces?
Outline

• Overview of MOOCHO

• Requirements for levels of optimization => Linear algebra Interfaces

• Vector reduction/transformation operators (RTOp)

• Timing experiments
• Compare
  – RTOp (all-at-once reduction (i.e. ISIS++ QMR solver))
    \[ \{ \alpha, \gamma, \xi, \rho, \epsilon \} \leftarrow \{ (x^T x)^{1/2}, (v^T v)^{1/2}, (w^T w)^{1/2}, w^T v, v^T t \} \]
  – Primitives (5 separate reductions)
    \[ \alpha \leftarrow (x^T x)^{1/2}, \gamma \leftarrow (v^T v)^{1/2}, \xi \leftarrow (w^T w)^{1/2}, \rho \leftarrow w^T v, \epsilon \leftarrow v^T \]
**RTOp vs. Primitives : Multiple Ops and Temporaries**

- **Compare**
  - **RTOp (all-at-once reduction)**
    \[
    \{ \max \alpha : x + \alpha d \geq \beta \} = \min \{ \max( (\beta - x_i)/d_i, 0 ), \text{ for } i = 1 \ldots n \} \rightarrow \alpha
    \]
  - **Primitives (5 temporaries, 6 vector operations)**
    \[
    -x_i \rightarrow u_i, \quad x_i + \beta \rightarrow v_i, \quad v_i / d_i \rightarrow w_i, \quad 0 \rightarrow y_i, \quad \max\{w_i,y_i\} \rightarrow z_i, \quad \min\{z_i,i=1\ldots n\} \rightarrow \alpha
    \]

* 1 processor (gcc 3.1 under Linux)
Parallel Scalability of MOOCHO

Scaleable example
NLP (m = n/2)

Variable reduction
range / null space
decomposition

\[ \min f(x) = \frac{1}{2} \sum_{i=1}^{n} x_i^2 \]
\[ s.t. \quad c_j(x) = x_j \left(x_{m+j} - 1\right) - 10x_{m+j} = 0 \quad j = 1, ..., n/2 \]

\[ A^T = \begin{bmatrix} C & N \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix} \]
\[ Z = \begin{bmatrix} -C^{-1}N \\ I \end{bmatrix} \]

Where is the parallel bottleneck?

Is it OO C++ or MPI? Answer => MPI

Serial overhead of MOOCHO (n=2, N_p=1)
\approx 0.41 \text{ milliseconds per rSQP iteration}

Overhead of MPI communication (N_p=4)
\approx 0.42 \text{ milliseconds per global reduction}

* Red Hat Linux cluster (4 nodes)
  - 2.0 GHz Intel P4 processors
  - MPICH 1.2.2.1

Parallel Speedup

<table>
<thead>
<tr>
<th>Np (number of processors)</th>
<th>n (global dim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20,000</td>
</tr>
<tr>
<td>2</td>
<td>200,000</td>
</tr>
<tr>
<td>3</td>
<td>2,000,000</td>
</tr>
<tr>
<td>4</td>
<td>4,000,000</td>
</tr>
</tbody>
</table>
Summary & Conclusions

**MOOCHO**

• MOOCHO is framework/library for large-scale NLP
• MOOCHO currently supports several active-set and interior-point SQP-related methods
• MOOCHO can be adapted to the application
• MOOCHO is fully scalable

**Interoperability of optimization applications and algorithms**

• Linear algebra interfaces are key to interoperability between applications and numerical algorithms
  
  Needed: Common application *linear algebra model*

  Needed: Common interface for vector operations *(RTOp)*
Availability & Future Work

Websites

• MOOCHO

• RTOp

Future Work

• Optimization algorithms/software
  – Full-space direct (Biegler & Laird)
  – Full-space iterative (help???)
  – Interior point (Biegler, Laird & Waechter)

• Applications
  – Sundance (ongoing)
  – Shape optimization (Euler equations with Edge/Reshape)
  – Shape optimization (Euler equations with Premo)
  – Circuit simulation (Xyce)
The End

Thank You!