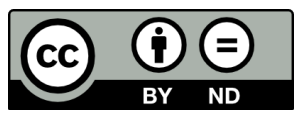
HPC considerations
for robust, reliable optimization codes

Joseph Young

Venue: Simula Research Laboratory, Fornebu, Norway
Presentation Number: PR 2015-03
Version: 1.0.0
Release Date: February 17, 2015
Revision Date: February 17, 2015

www.optimojoe.com



What do we mean by optimization?

In this presentation, we consider continuous, nonlinear optimization problems of the form

Unconstrained	**Equality Constrained**
$\min_{x \in X} \quad f(x)$	$\min_{x \in X} \quad f(x)$ $\text{st} \quad g(x) = 0$
Inequality Constrained	**Constrained**
$\min_{x \in X} \quad f(x)$ $\text{st} \quad h(x) \succeq 0$	$\min_{x \in X} \quad f(x)$ $\text{st} \quad g(x) = 0$ $\quad\quad h(x) \succeq 0$

What can we represent with these kinds of formulations?

Parameter estimation problems can be formulated as a constrained problem. For example,

$$\begin{aligned} \min_{\alpha \in L^2(\Omega), u \in H^1(\Omega \times [0,T])} \quad & \frac{1}{2}\|u - d\|^2 + R(\alpha) \\ \text{st} \quad & \frac{\partial}{\partial t} u - \nabla \cdot (\alpha \nabla u) = f \\ & \alpha \geq 0 \end{aligned}$$

matches a model of an object governed by the heat equation to experimental data where

- u - Simulated temperature measurements, K
- d - Experimental temperature measurements, K
- α - Thermal diffusivity, $m^2 \cdot s^{-1}$
- f - Temperature source, $K \cdot s^{-1}$
- R - Regularization, $m \cdot K$

What do optimization algorithms look like?

1. Initialize problem
2. While not converged
   - 2.1 Find new iterate
     - Newton's method
     - Nonlinear-CG
     - BFGS, SR1
     - Composite-step SQP
     - Primal-dual interior point method
   - 2.2 Check (globalize) new iterate
     - Line search
     - Trust region
   - 2.3 Move to new iterate
     - Update any applicable Lagrange multipliers
   - 2.4 Book keeping
     - Update any quasi-Newton information for BFGS and SR1
     - Store old gradient information for nonlinear-CG
     - Save any restart information

How to calculate unconstrained iterates

- **Newton's method**: Solve

$$\nabla^2 f(x)\partial x = -\nabla f(x)$$

for ∂x using a preconditioner $P_{\nabla^2 f(x)}$

- **Nonlinear-CG**

$$\partial x = -\nabla f(x) + \beta \partial x_{old}$$

where
  - Fletcher-Reeves: $\beta = \frac{\langle \nabla f(x), \nabla f(x) \rangle}{\langle \nabla f(x)_{old}, \nabla f(x)_{old} \rangle}$
  - Polak-Ribiere: $\beta = \frac{\langle \nabla f(x), \nabla f(x) - \nabla f(x)_{old} \rangle}{\langle \nabla f(x)_{old}, \nabla f(x)_{old} \rangle}$

- **BFGS**

$$\partial x = -B_k \nabla f(x)$$

where

$$\begin{aligned} B_{k+1} &= B_k - \frac{\langle B_k s_k, \cdot \rangle}{\langle B_k s_k, s_k \rangle} B_k s_k + \frac{\langle y_k, \cdot \rangle}{\langle y_k, s_k \rangle} y_k \\ s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \end{aligned}$$

How to calculate equality constrained iterates

- **Composite-step SQP**: Solve the system

$$\begin{aligned} \nabla f(x) + g'(x)^* y &= 0 \\ g(x) &= 0 \end{aligned}$$

in two steps by solving

$$\begin{aligned} \min_{\partial x_n \in X} \quad & \tfrac{1}{2}\|\partial x_n\|^2 \\ \text{st} \quad & g'(x)\partial x_n = -g(x) \end{aligned}$$

first for the normal step and then

$$Z^T(\nabla^2 f(x) + (g''(x)\cdot)^* y) Z \hat{\partial x_t} = -Z^T(\nabla f(x) + g'(x)^* y)$$

for the tangential step $\partial x_t = Z\hat{\partial x_t}$ where Z projects us onto the nullspace of $g'(x)$. In each step, use the preconditioner $P_{g'(x)g'(x)^*}$.

How to calculate inequality constrained iterates

- **Primal-dual interior point method**: Compute the iterates like we did above, but add

$$-\mu h'(x)^*(L(h(x))^{-1}e$$

to the gradient, add

$$h'(x)^*(L(h(x))^{-1}(h'(x)\cdot \circ z))$$

to the Hessian, and insure

$$x + \alpha \partial x \succeq 0$$

for some α where

  - $\circ$ - Jordan product (most of the time elementwise product)
  - $L(\cdot)^{-1}$ - Inverse of the the Jordan product
  - e - Jordan product identity

How to check (globalize) iterates

For some merit function ϕ,

- **Trust-region** - Check that

$$\phi(x) - \phi(x + \partial x) \geq \eta_1(\phi(x) - m(\partial x))$$

where

$$m(\partial x) = \frac{1}{2}\langle \nabla^2 \phi(x)\partial x, \partial x\rangle + \langle \nabla\phi(x), \partial x\rangle + \phi(x)$$

- **Line-search** - Check that

$$\begin{aligned} \phi(x + \alpha\partial x) \leq & \phi(x) + c_1\alpha\langle \nabla\phi(x), \partial x\rangle \\ \langle \nabla\phi(x + \alpha\partial x), \partial x\rangle \geq & c_2\langle \nabla\phi(x), \partial x\rangle \end{aligned}$$

where α denotes the result of the line search

Core optimization operations

Functions (and preconditioners)

- $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$
- $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$
- $h(x)$, $h'(x)\partial x$, $h'(x)^* z$

Algebra (unconstrained, equality constrained)

- $copy(x) \leftarrow x$
- $scal(\alpha, x) \leftarrow \alpha x$
- $axpy(\alpha, x, y) \leftarrow \alpha x + y$
- $innr(x, y) \leftarrow \langle x, y \rangle$

Jordan algebra (inequality constrained)

- $prod(x, y) \leftarrow x \circ y = L(x)y$
- $linv(x, y) \leftarrow L(x)^{-1} y$
- $id(x) \leftarrow e$ where $x \circ e = x$
- $srch(x, y) \leftarrow \arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$
- $barr(x) \leftarrow \phi(x)$ where $x \circ \nabla\phi(x) = e$

Opportunities for parallelism

In each of the above algorithms, we can parallelize

- **Functions** - $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$, $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$, $h(x)$, $h'(x)\partial x$, $h'(x)^* z$
- **Algebra** - *copy*, *scal*, *axpy*, *innr*
- **Jordan algebra** - *prod*, *linv*, *id*, *srch*, *barr*
- **Line search**

Should the user or optimization code control this parallelism?

Parallelism in the functions

Let us focus on our functions and their derivatives

- $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$
- $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$
- $h(x)$, $h'(x)\partial x$, $h'(x)^* z$

There are two kinds of parallelism

- **Computational parallelism** - computing these functions
- **Data parallelism** - storing the results in parallel

Most models based on mechanics require both

- Data on large meshes needs parallel storage (data parallelism)
- Computations on these meshes such as finite element matrix assembly or finite difference stencil application must be done in parallel (computation parallelism)

Optimization solver can **not** determine how to do this.
User must define both the computational and data parallelism for their functions.

Parallelism in the algebra

If we assume that the user must define both the data and computational parallelism for the functions, this means the user defines how to store x, y, and z, but these are manipulated by the algebraic operations:

Algebra

- $copy(x) \leftarrow x$
- $scal(\alpha, x) \leftarrow \alpha x$
- $axpy(\alpha, x, y) \leftarrow \alpha x + y$
- $innr(x, y) \leftarrow \langle x, y \rangle$

Jordan algebra

- $prod(x, y) \leftarrow x \circ y = L(x)y$
- $linv(x, y) \leftarrow L(x)^{-1}y$
- $id(x) \leftarrow e$ where $x \circ e = x$
- $srch(x, y) \leftarrow \arg\max\{\alpha \in \mathbb{R} : \alpha x + y \succeq 0, \alpha \geq 0\}$
- $barr(x) \leftarrow \phi(x)$ where $x \circ \nabla\phi(x) = e$

User must also define the computational and data parallelism for the algebra.

Parallelism in the line search

Most line search algorithms require a sequence of evaluations

$$\{\phi(x + \alpha_1 \partial x), \langle \nabla\phi(x + \alpha_1 \partial x), \partial x\rangle\},$$
$$\vdots$$
$$\{\phi(x + \alpha_m \partial x), \langle \nabla\phi(x + \alpha_m \partial x), \partial x\rangle\}$$

where ϕ denotes the merit function

- Evaluations can be done independently of one another
- Exact number of evaluations and where they occur depends on the line-search algorithm
- Can not expect our users to know the details of the line-search algorithms

Optimization code must be responsible for the parallelism of the line search.

Opportunities for parallelism

User

- **Functions** - $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$, $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$, $h(x)$, $h'(x)\partial x$, $h'(x)^* z$
- **Algebra** - *copy*, *scal*, *axpy*, *innr*
- **Jordan algebra** - *prod*, *linv*, *id*, *srch*, *barr*

Optimization code

- **Line search**

Generally, it's not worth parallelizing the line search because

- Amount of computational savings is minimal if the functions themselves are parallelized
- Only really pays off if
  - Line search does no gradient calculations
  - Evaluation of merit function massively cheaper than gradient
- Greatly complicates the optimization code because this is the **only** place the optimization code should control the parallelism and it's not clear what parallel method should be used (threads, MPI, etc.)

Final comments on parallelism

- All worthwhile places for parallelism defined by the user
- As long as we adhere to the abstractions
  - **Functions** - $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$, $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$, $h(x)$, $h'(x)\partial x$, $h'(x)^* z$
  - **Algebra** - *copy*, *scal*, *axpy*, *innr*
  - **Jordan algebra** - *prod*, *linv*, *id*, *srch*, *barr*

  the optimization code can
  - Use any form of data parallelism (MPI, grid, etc.)
  - Use any form of computational parallelism (MPI, OpenMP, GPU, threads, STM)
  - Immediately adapt to new forms of parallelism not yet conceived

Revisiting the heat equation

Let us simplify the parameter estimation problem based on the heat equation from before

$$\begin{aligned} \min_{\alpha \in L^2(\Omega), u \in H^1(\Omega \times [0,T])} \quad & \frac{1}{2}\|u - d\|^2 + R(\alpha) \\ \text{st} \quad & \frac{\partial}{\partial t} u - \nabla \cdot (\alpha \nabla u) = f \\ & \alpha \geq 0 \end{aligned}$$

into the 1-D problem

$$\begin{aligned} \min_{\alpha \in \mathbb{R}, u \in H^1([0,L] \times [0,T])} \quad & \frac{1}{2}\|u - d\|^2 + R(\alpha) \\ \text{st} \quad & \frac{\partial}{\partial t} u - \alpha \frac{\partial^2}{\partial x^2} u = 0 \\ & u(0,t) = a(t) \\ & u(L,t) = b(t) \\ & u(x,0) = f(x) \end{aligned}$$

Revisiting the heat equation

- Assume we have a solution operator ϕ that when given a thermal diffusivity, α, it solves the heat equation above
- This allows us to reformulate the above problem into the reduced space formulation

$$\min_{\alpha \in \mathbb{R}} \quad \frac{1}{2}\|\phi(\alpha) - d\|^2 + R(\alpha)$$

- For reference, the full space formulation is

$$\begin{aligned} \min_{\alpha \in \mathbb{R}, u \in H^1([0,L]\times[0,T])} \quad & \frac{1}{2}\|u - d\|^2 + R(\alpha) \\ \text{st} \quad & \frac{\partial}{\partial t}u - \alpha \frac{\partial^2}{\partial x^2}u = 0 \\ & u(0,t) = a(t) \\ & u(L,t) = b(t) \\ & u(x,0) = f(x) \end{aligned}$$

- At the moment, reduced space formulations are the de facto standard formulation for parameter estimation

Numerical difficulties in reduced space formulations

- Let us solve the heat equation with a finite difference method
- Use a forward-Euler method to discretize time and a central difference method to discretize space

$$u_{i,j+1} = \mu u_{i+1,j} + (1 - 2\mu)u_{i,j} + \mu u_{i-1,j}$$

where

$$\mu = \frac{\alpha \Delta t}{\Delta x^2}$$

- von Neumann stability requires

$$\Delta t \leq \frac{\Delta x^2}{2\alpha}$$

- Stability limit depends on the optimization variable α

Numerical difficulties in reduced space formulations

To recap

- Reduced space formulation for the heat equation is

$$\min_{\alpha \in \mathbb{R}} \quad \frac{1}{2}\|\phi(\alpha) - d\|^2 + R(\alpha)$$

where $\phi(\alpha)$ returns u that solves

$$\begin{aligned} \frac{\partial}{\partial t}u - \alpha \frac{\partial^2}{\partial x^2}u &= 0 \\ u(0,t) &= a(t) \\ u(L,t) &= b(t) \\ u(x,0) &= f(x) \end{aligned}$$

- In a finite difference method, von Neumann stability requires

$$\Delta t \leq \frac{\Delta x^2}{2\alpha}$$

- Above formulation has no bound on α

Numerical difficulties in reduced space formulations

What happens when we violate the stability limit?

- Sometimes we get a weird solution
- Most of the time, the solver return NaNs

How should do we handle these errors?

- Ideally, we should add an inequality constraint to bound our material
- Absent an explicitly formulated bound, we must handle NaNs
  - Arise during the *find new iterate* portion of optimization
  - As long as we
    - Start with a stable set of parameters
    - Globalization rejects steps with NaNs

    our algorithms can handle instability by reducing the step
  - Methodology akin to a poor man's projection method, except we don't need explicit bounds and it doesn't work as well

Practical issues when running large optimization solves

Optimization solves can fail for a variety of issues

- Parallel cluster goes down due to a corrupted core
- Parallel cluster goes down for software upgrades
- Run terminated due exceeding allocated time allowance
- Parallel cluster goes down due to someone running a thousand single processor runs on the login node where each run reads directly from the networked disk rather than scratch space[1]

In each of these cases, it's paramount that we do not lose spent computation on our optimization solve

[1] Used to happen weekly

What is a restart?

- **Restarts** (also known as checkpoint/restart) write out the entire state of the optimization algorithm
- Generally, state of the algorithm written once per iteration
- In the event of a crash, we restart the computation at the last successful iteration

Anatomy of a restart

Optimization algorithms generally consist of

- **Real** numbers
- **Natural** numbers (nonnegative integers)
- **Parameters** (enumerated types convertible to strings)
- **Vectors** (vector spaces X,Y,Z)
  - X - domain of the optimization
  - Y - codomain of the equality constraints
  - Z - codomain of the inequality constraints

For each of these four classes of elements, we must

- Be able to iterate over each
- Know how to serialize each

Should the user or optimization code control this serialization?

Parallelism considerations

From our section on parallelism, user defines how vectors are stored. Therefore,

User

- **Vectors**

Optimization code

- **Reals**
- **Naturals**
- **Parameters**

Since user defines how vectors are serialized, vector serialization can be done in parallel

Parallelism considerations

How do we write the serialized elements to disk?

- Do we write to disk or over a network?
- In a single program/multiple data (SPMD) situation such as with MPI, who writes the final restart file?

Optimization solver can not know how to do this most effectively.
User must define how serialized information is read from and written to disk

Can we do better?

- In a reduced space formulation, each Hessian-vector product requires one forward and one adjoint solve
- Each forward and adjoint solve requires the solution of a PDE, which is likely expensive
- Inexact Newton methods may require tens to hundreds of iterations before the iteration completes
- Important not to lose this information

Ideally, each Krylov iteration in an inexact Newton method should also be serialized to disk

Summary

- Abstracting algorithms on
  - Functions - $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)\partial x$, $P_{\nabla^2 f(x)}\partial x$, $g(x)$, $g'(x)\partial x$, $g'(x)^* y$, $(g''(x)\partial x)^* y$, $P_{g'(x)g'(x)^*}\partial y$, $h(x)$, $h'(x)\partial x$, $h'(x)^* z$
  - Algebra - *copy*, *scal*, *axpy*, *innr*
  - Jordan algebra - *prod*, *linv*, *id*, *srch*, *barr*

  allows arbitrary forms of parallelism
- Handling NaNs properly makes codes robust toward implicit stability requirements of PDE constrained optimization
  - Start with a stable set of parameters
  - Globalization rejects steps with NaNs
- Restarts important to recovering from system crashes
  - Serialize state of the optimization every iteration
  - Ideally, serialize the state of the Krylov methods every iteration
  - User defines how to serialize vectors and how to write serialization to disk

COMPUTATIONAL OPTIMIZATION