

An Object Oriented Architecture for Nonlinear Constrained Optimization

Edward Michael Gertz
University of Wisconsin, Madison

January 15, 2003

Collaborative work with:

Philip E. Gill and Joshua Griffin, University of California, San Diego.
Stephen J. Wright, University of Wisconsin, Madison.

E. Michael Gertz <gertz@mcs.anl.gov>

Thanks

I'd like to thank:

- ▶ The University of Wisconsin, Madison
- ▶ Northwestern University
- ▶ Argonne National Laboratory and the Department of Energy
- ▶ The National Science Foundation
- ▶ My collaborators: Philip E. Gill, Joshua Griffin and Stephen J. Wright

Thanks

I'd like to thank:

- ▶ The University of Wisconsin, Madison
- ▶ [Northwestern University](#)
- ▶ Argonne National Laboratory and the Department of Energy
- ▶ The National Science Foundation
- ▶ My collaborators: Philip E. Gill, Joshua Griffin and Stephen J. Wright

Thanks

I'd like to thank:

- ▶ The University of Wisconsin, Madison
- ▶ Northwestern University
- ▶ Argonne National Laboratory and the Department of Energy
- ▶ The National Science Foundation
- ▶ My collaborators: Philip E. Gill, Joshua Griffin and Stephen J. Wright

Thanks

I'd like to thank:

- ▶ The University of Wisconsin, Madison
- ▶ Northwestern University
- ▶ Argonne National Laboratory and the Department of Energy
- ▶ The National Science Foundation
- ▶ My collaborators: Philip E. Gill, Joshua Griffin and Stephen J. Wright

Thanks

I'd like to thank:

- ▶ The University of Wisconsin, Madison
- ▶ Northwestern University
- ▶ Argonne National Laboratory and the Department of Energy
- ▶ The National Science Foundation
- ▶ My collaborators: Philip E. Gill, Joshua Griffin and Stephen J. Wright

Outline

I will discuss:

- ▶ the algorithm and why it is amenable to an object-oriented implementation;

Outline

I will discuss:

- ▷ the algorithm and why it is amenable to an object-oriented implementation;
- ▷ the structure of the code; and

Outline

I will discuss:

- ▷ the algorithm and why it is amenable to an object-oriented implementation;
- ▷ the structure of the code; and
- ▷ a technical issue in object-oriented programming that arised due to this design.

Outline

I will discuss:

- ▷ the algorithm and why it is amenable to an object-oriented implementation;
- ▷ the structure of the code; and
- ▷ a technical issue in object-oriented programming that arised due to this design.

I imagine each member of the audience will be interested in two of these topics.

Notation

Consider the non-linear program

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & c(x) = 0, \quad d(x) \geq 0. \end{array}$$

Let

- ▷ $g(x) = \nabla f(x)$
- ▷ $H = H(x, u, v) = \nabla_x^2 (f(x) - c(x)^T u - d(x)^T v)$.
- ▷ J_c and J_d are the Jacobians of $c(x)$ and $d(x)$ respectively.
- ▷ $C = \text{diag}(c)$, $D = \text{diag}(d)$ and $V = \text{diag}(v)$.

The Primal-Dual System

The primal-dual system is equivalent to the symmetric system

$$\left(\begin{array}{c|cc} H & J_c^T & J_d^T \\ \hline J_c & -\rho I & 0 \\ J_d & 0 & -V^{-1}D \end{array} \right) \begin{pmatrix} \Delta x \\ -\Delta u \\ -\Delta v \end{pmatrix} = - \begin{pmatrix} g - J_c^T u - J_d^T v \\ c + \rho u \\ D - \mu V^{-1}e \end{pmatrix}$$

This system be rewritten using the notation

$$\begin{pmatrix} H & J^T \\ J & -W \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta z \end{pmatrix} = - \begin{pmatrix} g - J^T z \\ W(z - \zeta) \end{pmatrix},$$

where $W = \begin{pmatrix} \rho I & 0 \\ 0 & V^{-1}D \end{pmatrix}$, $z = \begin{pmatrix} u \\ v \end{pmatrix}$ and $\zeta = \begin{pmatrix} -c/\rho \\ \mu D^{-1}e \end{pmatrix}$

The Primal-Dual System

The primal-dual system is equivalent to the symmetric system

$$\left(\begin{array}{c|cc} H & J_c^T & J_d^T \\ \hline J_c & -\rho I & 0 \\ J_d & 0 & -V^{-1}D \end{array} \right) \begin{pmatrix} \Delta x \\ -\Delta u \\ -\Delta v \end{pmatrix} = - \begin{pmatrix} g - J_c^T u - J_d^T v \\ c + \rho u \\ D - \mu V^{-1} e \end{pmatrix}$$

This system be rewritten using the notation

$$\begin{pmatrix} H & J^T \\ J & -W \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta z \end{pmatrix} = - \begin{pmatrix} g - J^T z \\ W(z - \zeta) \end{pmatrix},$$

where $W = \begin{pmatrix} \rho I & 0 \\ 0 & V^{-1}D \end{pmatrix}$, $z = \begin{pmatrix} u \\ v \end{pmatrix}$ and $\zeta = \begin{pmatrix} -c/\rho \\ \mu D^{-1} e \end{pmatrix}$

The Primal-Dual System

The primal-dual system is equivalent to the symmetric system

$$\left(\begin{array}{c|cc} H & J_c^T & J_d^T \\ \hline J_c & -\rho I & 0 \\ J_d & 0 & -V^{-1}D \end{array} \right) \begin{pmatrix} \Delta x \\ -\Delta u \\ -\Delta v \end{pmatrix} = - \begin{pmatrix} g - J_c^T u - J_d^T v \\ c + \rho u \\ D - \mu V^{-1}e \end{pmatrix}$$

This system be rewritten using the notation

$$\begin{pmatrix} H & J^T \\ J & -W \end{pmatrix} \begin{pmatrix} \Delta x \\ -\Delta z \end{pmatrix} = - \begin{pmatrix} g - J^T z \\ W(z - \zeta) \end{pmatrix},$$

where $W = \begin{pmatrix} \rho I & 0 \\ 0 & V^{-1}D \end{pmatrix}$, $z = \begin{pmatrix} u \\ v \end{pmatrix}$ and $\zeta = \begin{pmatrix} -c/\rho \\ \mu D^{-1}e \end{pmatrix}$

Forsgren-Gill merit function

Forsgren and Gill (1998) define the function

$$\begin{aligned} \mathcal{M}_{\mu,\rho}(x, u, v) &= f(x) + \frac{1}{2\rho}(\|c(x)\|^2 + \|c(x) + \rho u\|)^2 - \mu \sum_i \log d_i(x) \\ &\quad - \mu \sum_i \left\{ \log \left(\frac{v_j d_i(x)}{\mu} \right) + \left(1 - \frac{v_j d_i(x)}{\mu} \right) \right\}, \end{aligned}$$

and show that minimizers of this function satisfy the perturbed optimality conditions.

This function is the standard barrier-penalty function, plus additional terms involving the Lagrange multipliers u and v .

Properties of the Merit function

- ▶ The gradient of the merit function is $\begin{pmatrix} g - J^T(2\zeta - z) \\ W(z - \zeta) \end{pmatrix}$.

Properties of the Merit function

- ▶ The gradient of the merit function is $\begin{pmatrix} g - J^T(2\zeta - z) \\ W(z - \zeta) \end{pmatrix}$.
- ▶ The second derivative of the merit function is approximated by

$$\begin{pmatrix} H + 2J^TW^{-1}J & J^T \\ J & W \end{pmatrix}$$

and this approximation is exact on the trajectory.

Properties of the Merit function

- ▶ The gradient of the merit function is $\begin{pmatrix} g - J^T(2\zeta - z) \\ W(z - \zeta) \end{pmatrix}$.
- ▶ The second derivative of the merit function is approximated by

$$\begin{pmatrix} H + 2J^TW^{-1}J & J^T \\ J & W \end{pmatrix}$$

and this approximation is exact on the trajectory.

- ▶ The approximate Newton equations are equivalent to the primal-dual equations.

A Trust-Region Approach

Generate trial steps by minimizing the quadratic model

$$\begin{aligned} \mathcal{Q}(p, q) = & \begin{pmatrix} p^T & q^T \end{pmatrix} \begin{pmatrix} g - J^T(2\zeta - z) \\ W(z - \zeta) \end{pmatrix} \\ & + \frac{1}{2} \begin{pmatrix} p^T & q^T \end{pmatrix} \begin{pmatrix} H + 2J^TW^{-1}J & J^T \\ J & W \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} \end{aligned}$$

subject to a restriction on the length of the step

$$p^TMp + q^TNq \leq \delta^2$$

Where $\delta > 0$ and M and N are symmetric, positive definite matrices.

Solving the trust-region subproblem

We repeatedly solve systems of the form

$$\begin{pmatrix} H + 2J^T W^{-1} J + \sigma M & J^T \\ J & W + \sigma N \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = - \begin{pmatrix} g - J^T(2\zeta - z) \\ W(z - \zeta) \end{pmatrix}$$

These systems can be manipulated into the form

$$\begin{pmatrix} H + \sigma M & J^T \\ J & -\widehat{W}(\sigma) \end{pmatrix} \begin{pmatrix} p \\ -\hat{q} \end{pmatrix} = - \begin{pmatrix} g - J^T z \\ W(z - \zeta) \end{pmatrix}$$

where

$$\hat{q} = (I + 2\sigma W^{-1} N)q \quad \text{and} \quad \widehat{W}(\sigma) = (W + \sigma N)(I + 2\sigma W^{-1} N)^{-1}.$$

Summary of Theory

- ▶ Solutions to the perturbed optimality conditions can be found by minimizing a certain function in both x and z .

Summary of Theory

- ▶ Solutions to the perturbed optimality conditions can be found by minimizing a certain function in both x and z .
- ▶ An approximate Newton system for minimizing this merit function is equivalent to the primal-dual system.

Summary of Theory

- ▶ Solutions to the perturbed optimality conditions can be found by minimizing a certain function in both x and z .
- ▶ An approximate Newton system for minimizing this merit function is equivalent to the primal-dual system.
- ▶ Trust-region techniques may be used to minimize the merit function.

Summary of Theory

- ▶ Solutions to the perturbed optimality conditions can be found by minimizing a certain function in both x and z .
- ▶ An approximate Newton system for minimizing this merit function is equivalent to the primal-dual system.
- ▶ Trust-region techniques may be used to minimize the merit function.
- ▶ The matrices that must be factored to solve the trust-region subproblem have the same structure as the primal-dual system. No special factorization is required.

Accepting an Idea

Accepting an Idea

- ▶ It's impossible.

Accepting an Idea

- ▷ It's impossible.
- ▷ Maybe it's possible, but it's weak and uninteresting.

Accepting an Idea

- ▷ It's impossible.
- ▷ Maybe it's possible, but it's weak and uninteresting.
- ▷ It is true and I told you so.

Accepting an Idea

- ▷ It's impossible.
- ▷ Maybe it's possible, but it's weak and uninteresting.
- ▷ It is true and I told you so.
- ▷ I thought of it first.

Accepting an Idea

- ▷ It's impossible.
- ▷ Maybe it's possible, but it's weak and uninteresting.
- ▷ It is true and I told you so.
- ▷ I thought of it first.
- ▷ How could it be otherwise.

General Observations

- ▶ In simulation-based optimization, it is important to allow domain experts to supply specialized data-structures and routines.

General Observations

- ▶ In simulation-based optimization, it is important to allow domain experts to supply specialized data-structures and routines.
- ▶ Small classes, with limited functionality and unspecified data structures, are easiest to specialize, or completely replace.

General Observations

- ▶ In simulation-based optimization, it is important to allow domain experts to supply specialized data-structures and routines.
- ▶ Small classes, with limited functionality and unspecified data structures, are easiest to specialize, or completely replace.
- ▶ Concrete classes with extensive functionality are convenient to use.

General Observations

- ▶ In simulation-based optimization, it is important to allow domain experts to supply specialized data-structures and routines.
- ▶ Small classes, with limited functionality and unspecified data structures, are easiest to specialize, or completely replace.
- ▶ Concrete classes with extensive functionality are convenient to use.

These observations suggest a layered design, in which simple abstract classes are built from more complex concrete classes.

General Observations

- ▶ In simulation-based optimization, it is important to allow domain experts to supply specialized data-structures and routines.
- ▶ Small classes, with limited functionality and unspecified data structures, are easiest to specialize, or completely replace.
- ▶ Concrete classes with extensive functionality are convenient to use.

These observations suggest a layered design, in which simple abstract classes are built from more complex concrete classes.

Designs that specify a limited number of primitives from which a default implementation may be built are helpful (but beyond this talk.)

Three Layers to lotr

lotr has a layered design.

Solver Layer implements the high-level algorithm and heuristics.

Each layer of lotr is defined entirely in terms of operations from the layer below it.

Three Layers to lotr

lotr has a layered design.

Solver Layer implements the high-level algorithm and heuristics.



Problem Formulation Layer models a problem in its natural form.

Each layer of lotr is defined entirely in terms of operations from the layer below it.

Three Layers to lotr

lotr has a layered design.

Solver Layer implements the high-level algorithm and heuristics.



Problem Formulation Layer models a problem in its natural form.



Linear Algebra Layer manipulates basic linear algebra objects.

Each layer of lotr is defined entirely in terms of operations from the layer below it.

Solver Layer

The solver layer implements a primal-dual interior point trust-region algorithm.

This layer only concerned with the highest-level algorithms and heuristics. Other details are taken care of by the lower layers.

The trust-region algorithm makes it possible to describe the solver abstractly.

Customization of the solver layer is available through delegation. For instance the termination tests may be delegated to user-supplied piece of code.

Problem Formulation Layer

Many problems, for instance problems in trajectory optimization, have exploitable structure. The problem formulation layer:

- ▷ is a bridge between the natural model of a problem and an abstract NLP formulation that the solver layer understands.
- ▷ exploits problem structure to optimize for memory or speed;
- ▷ performs high-level linear algebra operations, such as block elimination, but invokes the linear algebra layer to perform lower-level optimizations.

Linear Algebra Layer

The linear algebra layer performs basic linear algebra operations such as matrix multiplications and linear system solves.

The layer is primarily an adaptor between the lotr code and existing packages that provide the required functionality.

One can develop a solver customized to a computing environment by invoking a particular implementation of the linear algebra layer.

It is possible to embed a solver in a program by developing a customized linear algebra layer.

Covariant Specialization

A layer consists of classes that implement an abstract interface, but are designed to work well together.

Covariant Specialization

A layer consists of classes that implement an abstract interface, but are designed to work well together.

To interoperate, it is usually necessary to invoke capabilities not in the abstract interface. For object-oriented code, this will require downcasting.

Covariant Specialization

A layer consists of classes that implement an abstract interface, but are designed to work well together.

To interoperate, it is usually necessary to invoke capabilities not in the abstract interface. For object-oriented code, this will require downcasting.

Techniques of generic programming, or template-based specialization can be used to avoid downcasting, but are not without a price.

The Debate

Those who feel strongly on the issue seem to divide into two camps, which are locked in a fight to the death.

The Debate

Those who feel strongly on the issue seem to divide into two camps, which are locked in a fight to the death.

Camp 1 Downcasting is sometimes necessary, and usually harmless. If you are too worried about it you are an idiot.

The Debate

Those who feel strongly on the issue seem to divide into two camps, which are locked in a fight to the death.

Camp 1 Downcasting is sometimes necessary, and usually harmless. If you are too worried about it you are an idiot.

Camp 2 Downcasting violates static type checking, which is critical to good code development. Therefore if you don't use templates to avoid this issue, you are an idiot.

The Debate

Those who feel strongly on the issue seem to divide into two camps, which are locked in a fight to the death.

Camp 1 Downcasting is sometimes necessary, and usually harmless. If you are too worried about it you are an idiot.

Camp 2 Downcasting violates static type checking, which is critical to good code development. Therefore if you don't use templates to avoid this issue, you are an idiot.

Because I am sometimes contrary (but hopefully not an idiot) I take a different view. I have observed that downcasting can have benefits.

Workflow

We can use downcasting to enable a "write and refactor" style of development.

Workflow

We can use downcasting to enable a "write and refactor" style of development.

- ▶ Develop a reference implementation of a problem formulation quickly, by downcasting freely and using the capabilities of concrete classes.

Workflow

We can use downcasting to enable a "write and refactor" style of development.

- ▶ Develop a reference implementation of a problem formulation quickly, by downcasting freely and using the capabilities of concrete classes.
- ▶ Develop tests based on this formulation.

Workflow

We can use downcasting to enable a "write and refactor" style of development.

- ▶ Develop a reference implementation of a problem formulation quickly, by downcasting freely and using the capabilities of concrete classes.
- ▶ Develop tests based on this formulation.
- ▶ Replace concrete operations and classes with appropriate abstract versions.

Workflow

We can use downcasting to enable a "write and refactor" style of development.

- ▶ Develop a reference implementation of a problem formulation quickly, by downcasting freely and using the capabilities of concrete classes.
- ▶ Develop tests based on this formulation.
- ▶ Replace concrete operations and classes with appropriate abstract versions.

Of course, if you were smart enough, you would get the interfaces right in the first place, and not need to do this.