A Smooth Exact Penalty Function for Nonlinear Optimization

Ron Estrin
ICME, Stanford University

Sandia National Lab
Feb. 21, 2018

Joint Work with Michael Friedlander, Dominique Orban, and Michael Saunders

Penalty methods for Nonlinear Programming

Equality constrained nonlinear program

$$\begin{array}{ll} \min\limits_{x} & f(x) \\ \text{s.t.} & c(x) = 0, \end{array}$$

with n variables, $m \leq n$ constraints and $f, c \in C^2$.

Plethora of methods available, but still an active area of research!

Methods for NLP often complicated:

- involve complicated heuristics to trade off optimality vs. feasibility
- feasibility restoration phases required

Penalty methods

Add some measure of constraint violation in objective

- **Quadratic penalty**

$$\min_x f(x) + \frac{\sigma_k}{2}\|c(x)\|_2^2$$

  - Perturbs the solution.
  - Need to solve sequence of problems with $\sigma_k \to \infty$.

Penalty methods

Add some measure of constraint violation in objective

- **Quadratic penalty**

$$\min_x f(x) + \frac{\sigma_k}{2} \|c(x)\|_2^2$$

  - Perturbs the solution.
  - Need to solve sequence of problems with $\sigma_k \to \infty$.

- ℓ_1 **penalty**

$$\min_x f(x) + \sigma \|c(x)\|_1$$

  - Non-smooth.

Penalty methods

Add some measure of constraint violation in objective

- **Augmented Lagrangian method**

$$\min_x f(x) + \lambda_k^T c(x) + \frac{\sigma_k}{2}\|c(x)\|_2^2$$

  - Need to solve sequence of problems.

Penalty methods

Add some measure of constraint violation in objective

- **Augmented Lagrangian method**

$$\min_x f(x) + \lambda_k^T c(x) + \frac{\sigma_k}{2}\|c(x)\|_2^2$$

  - Need to solve sequence of problems.

- Would like exact penalty function which is smooth...

Fletcher's Penalty Function

Primal only Lagrangian:

$$L(x, y) = f(x) - y^T c(x)$$

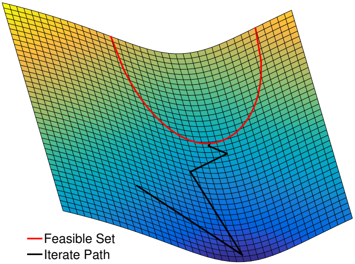
Fletcher's penalty function:

$$\begin{aligned}\phi_\sigma(x) &= f(x) - c(x)^T y_\sigma(x)\\ y_\sigma(x) &= \arg\min_y \frac{1}{2}\|g - Ay\|_2^2 + \sigma c(x)^T y\end{aligned}$$

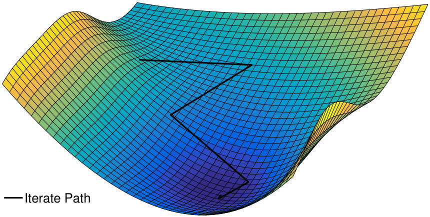
Notation:

$$g = \nabla f(x), \qquad A = \begin{bmatrix} \nabla c(x) \end{bmatrix}, \qquad Y_\sigma = \begin{bmatrix} \nabla y_\sigma(x) \end{bmatrix}.$$

An Illustration



(a) ‘hs007’ with feasible set.



(b) ϕ_σ for problem ‘hs007’.

Derivatives

Gradient of penalty function:

$$\begin{aligned}\nabla\phi_\sigma(x) &= g - Ay_\sigma - Y_\sigma c \\ &= g_\sigma - Y_\sigma c\end{aligned}$$

Hessian of penalty function:

$$\begin{aligned}\nabla^2\phi_\sigma(x) &= H - \sum_{i=1}^{m}(y_\sigma)_i H_i - AY_\sigma^T - Y_\sigma A^T - \nabla[Y_\sigma(\cdot)c] \\ &= H_\sigma - AY_\sigma^T - Y_\sigma A^T - \nabla[Y_\sigma(\cdot)c]\end{aligned}$$

Optimality Conditions

(x_*, y_*) is first-order KKT point:

$$c(x_*) = 0, \qquad g(x_*) - A(x_*)y_* = 0.$$

Optimality Conditions

(x_*, y_*) is first-order KKT point:

$$c(x_*) = 0, \qquad g(x_*) - A(x_*)y_* = 0.$$

Gradient of penalty function:

$$\nabla\phi_\sigma(x) = g - Ay_\sigma - Y_\sigma c.$$

Then $y_* = y_\sigma(x_*)$ and $\nabla\phi_\sigma(x_*) = 0$

$\implies$ **1st order KKT points** are **stationary points** of ϕ_σ.

Optimality Conditions

(x_*, y_*) is second-order KKT point:

$$d^T \nabla^2_{xx} L(x_*, y_*) d \geq 0, \qquad \text{for } d \text{ such that } A(x_*)^T d = 0$$

Optimality Conditions

(x_*, y_*) is second-order KKT point:

$$d^T \nabla^2_{xx} L(x_*, y_*) d \geq 0, \qquad \text{for } d \text{ such that } A(x_*)^T d = 0$$

Hessian of Penalty function:

$$\begin{aligned} \nabla^2 \phi_\sigma(x_*) &= H_\sigma - AY_\sigma^T - Y_\sigma A^T - \nabla[Y_\sigma(x)c] \\ &= H_\sigma - H_\sigma P_A - P_A H_\sigma + 2\sigma P_A \\ &= \bar{P}_A H_\sigma \bar{P}_A - P_A H_\sigma P_A + 2\sigma P_A \end{aligned}$$

Projectors: $\qquad P_A = A(A^T A)^{-1} A^T, \qquad \bar{P}_A = I - P_A.$

Optimality Conditions

Hessian of Penalty function:

$$\nabla^2 \phi_\sigma(x_*) = \bar{P}_A H_\sigma \bar{P}_A - P_A H_\sigma P_A + 2\sigma P_A$$

Then $\nabla^2 \phi_\sigma(x_*) \succeq 0$ if $\sigma \geq \sigma_* = \frac{1}{2}\|P_A H_\sigma P_A\|_2$

$\implies$ **2nd order KKT points** are **minimizers** of ϕ_σ for $\sigma \geq \sigma_*$.

$\phi_\sigma(x)$ is a smooth, exact penalty function.

If σ is chosen large enough, it is enough to minimize $\phi_\sigma(x)$ once to obtain KKT point to original NLP.

- (Weakly-exact: spurious minima rare but still possible)

How to evaluate function/gradient/products with (approx.) Hessian?

Function Evaluation

Multiplier estimate:

$$y_\sigma(x) = \arg\min_y \frac{1}{2}\|g - Ay\|_2^2 + \sigma c(x)^T y$$

which is solved by

$$A^T A y_\sigma = A^T g - \sigma c$$

or equivalently,

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} g_\sigma \\ y_\sigma \end{bmatrix} = \begin{bmatrix} g \\ \sigma c \end{bmatrix}$$

Products with Y_σ

First, Jacobian of y_σ:

$$Y_\sigma = (H_\sigma - \sigma I)A(A^TA)^{-1} + S(x, g_\sigma)^T(A^TA)^{-1}$$

where

$$S(x, v)u = \begin{bmatrix} v^T H_1 u \\ \vdots \\ v^T H_m u \end{bmatrix}, \qquad S(x, v)^T u = \sum_{i=1}^{m} u_i H_i v$$

(Notice that $S(x_*, g_\sigma) = 0$ since $g_\sigma = 0$ at solution).

Products with Y_σ

$$
\begin{aligned}
Y_\sigma u &= (H_\sigma - \sigma I) A (A^T A)^{-1} u + S(x, g_\sigma)^T (A^T A)^{-1} u \\
&= (H_\sigma - \sigma I)(-w) + \sum_{i=1}^{m} v_i H_i g_\sigma
\end{aligned}
$$

where

$$
\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ -u \end{bmatrix}.
$$

Products with Y_σ^T

$$
\begin{aligned}
Y_\sigma^T u &= (A^T A)^{-1} A^T (H_\sigma - \sigma I) u + (A^T A)^{-1} S(x, g_\sigma) u \\
&= v
\end{aligned}
$$

where

$$
\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} (H_\sigma - \sigma I) u \\ -S(x, g_\sigma) u \end{bmatrix}.
$$

Gradient Computation

Gradient:

$$\nabla \phi_\sigma = g_\sigma - Y_\sigma c$$

Need to solve one additional augmented system.

Hessian products

Hessian product:

$$
\begin{aligned}
\nabla^2 \phi_\sigma(x_*) v &= H_\sigma v - A Y_\sigma^T v - Y_\sigma A^T v - \nabla[Y_\sigma(x) c] v \\
&\approx H_\sigma v - A Y_\sigma^T v - Y_\sigma A^T v \qquad \text{(remove third derivatives)} \\
&\approx H_\sigma v - P_A H_\sigma v - H_\sigma P_A v + 2\sigma P_A v
\end{aligned}
$$

Can further approximate by removing terms which are zero at solution.

Two augmented system solves per product.

Hessian products

Hessian product:

$$\begin{aligned}\nabla^2\phi_\sigma(x_*)v &= H_\sigma v - AY_\sigma^T v - Y_\sigma A^T v - \nabla[Y_\sigma(x)c]v \\ &\approx H_\sigma v - AY_\sigma^T v - Y_\sigma A^T v \qquad \text{(remove third derivatives)} \\ &\approx H_\sigma v - P_A H_\sigma v - H_\sigma P_A v + 2\sigma P_A v\end{aligned}$$

Can further approximate by removing terms which are zero at solution.

Two augmented system solves per product.

Fletcher showed that even with these approximations, **quadratic convergence** can still be retained.

"Algorithm"

$\phi_\sigma(x)$ is smooth and exact.

"Algorithm"

$\phi_\sigma(x)$ is smooth and exact.

Can evaluate function, gradient and approximate Hessian products.

“Algorithm”

$\phi_\sigma(x)$ is smooth and exact.

Can evaluate function, gradient and approximate Hessian products.

Pick your favourite unconstrained minimizer, and done!

"Algorithm"

$\phi_\sigma(x)$ is smooth and exact.

Can evaluate function, gradient and approximate Hessian products.

Pick your favourite unconstrained minimizer, and done!

The end! Questions?

"Algorithm"

$\phi_\sigma(x)$ is smooth and exact.

Can evaluate function, gradient and approximate Hessian products.

Pick your favourite unconstrained minimizer, and done!

The end! Questions?
...Not quite

Practicalities

A few notes and observations:

Practicalities

A few notes and observations:

- Solving augmented system (direct vs. iterative)

Practicalities

A few notes and observations:

- Solving augmented system (direct vs. iterative)
- Singular Jacobians (regularization)

Practicalities

A few notes and observations:

- Solving augmented system (direct vs. iterative)
- Singular Jacobians (regularization)
- Trust-region $>$ linesearch (indefinite Hessians)

Practicalities

A few notes and observations:

- Solving augmented system (direct vs. iterative)
- Singular Jacobians (regularization)
- Trust-region $>$ linesearch (indefinite Hessians)
- Adjusting penalty parameter, initial points, unboundedness...

Solving the Augmented System

- Sparse LDL
- Semi-normal equations (Q-less QR) + iterative refinement
- Iterative methods
  - Efficient with good preconditioners (I've heard these near-optimal preconditioners for PDE's are popular...)
  - Inexact function/gradient evaluation controlled via iterative solver tolerance

Solving the Augmented System: LSLQ/LNLQ

Iterative methods for least-squares (LSLQ) or least-norm problems (LNLQ).

- Equivalent to SYMMLQ on normal equations.
- Given $\sigma_{est} \geq \sigma_{\min}(A)$, can monitor $\|w_* - w_k\|$ and $\|v_* - v\|$.
- For near-optimal preconditioners, where $A^T = \begin{bmatrix} A_u^T & A_z^T \end{bmatrix}$ and $\mathcal{P} = A_u^T A_u$, then $\sigma_{\min}\left(\mathcal{P}^{-1} A^T A\right) \geq 1$.

Regularization

If A is singular, the ϕ_σ can be undefined.
Regularize least-squares problem:

$$\begin{aligned}
\phi_{\sigma\delta}(x) &= f(x) - c(x)^T y_{\sigma\delta}(x) \\
y_{\sigma\delta}(x) &= \arg\min_y \frac{1}{2} \left\| \begin{bmatrix} g \\ 0 \end{bmatrix} - \begin{bmatrix} A \\ \delta I \end{bmatrix} y \right\|_2^2 + \sigma c(x)^T y
\end{aligned}$$

Only change: augmented system becomes

$$\begin{bmatrix} I & A \\ A^T & -\delta^2 I \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Regularization

Assume that LICQ holds at x_*.

- $\delta > 0$ perturbs the solution.

Regularization

Assume that LICQ holds at x_*.

- $\delta > 0$ perturbs the solution.
- Want $\delta \to 0$ to recover exact solution.

Regularization

Assume that LICQ holds at x_*.

- $\delta > 0$ perturbs the solution.
- Want $\delta \to 0$ to recover exact solution.
- Want to retain superlinear/quadratic convergence.

Regularization

Assume that LICQ holds at x_*.

- $\delta > 0$ perturbs the solution.
- Want $\delta \to 0$ to recover exact solution.
- Want to retain superlinear/quadratic convergence.

---

for k=1,2,... **do**
  $\delta_k \leftarrow \max\left\{\delta_{k-1}^2, \|\nabla \phi_{\sigma\delta_{k-1}}(x_k)\|\right\}$
  Get x_{k+1} from one step on $\phi_{\sigma\delta_k}(x_k)$
end for

If quadratically convergent method used, entire method above remains quadratically convergent.

Inequality Constraints

Consider problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0, \\ & x \geq 0. \end{aligned}$$

Inequality Constraints

Consider problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0, \\ & x \geq 0. \end{aligned}$$

Modify Fletcher's penalty function to

$$\begin{aligned} \min_{x} \quad & \phi_\sigma(x) = f(x) - c(x)^T y_\sigma(x) \\ \text{s.t.} \quad & x \geq 0 \\ & y_\sigma(x) = \arg\min_{y} \frac{1}{2} \|g - Ay\|_X^2 + \sigma c(x)^T y \end{aligned}$$

Inequality Constraints

Consider problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0, \\ & x \geq 0. \end{aligned}$$

Modify Fletcher's penalty function to

$$\begin{aligned} \min_x \quad & \phi_\sigma(x) = f(x) - c(x)^T y_\sigma(x) \\ \text{s.t.} \quad & x \geq 0 \\ & y_\sigma(x) = \arg\min_y \frac{1}{2}\|g - Ay\|_{X}^2 + \sigma c(x)^T y \end{aligned}$$

- Smoothness holds for $x > 0$ (use interior method)
- Exactness still holds

Inverse Poisson Problem (Thanks Drew!)

$$\begin{aligned} \min_{u,z} \quad & \frac{1}{2}\int_\Omega (u(x) - \bar{u}(x))^2 dx + \frac{\alpha}{2}\int_\Omega z(x)^2 dx \\ \text{s.t} \quad & -\nabla\cdot(z\nabla u) = f, \qquad \text{in } \Omega \\ & u = 0, \qquad \text{on } \partial\Omega \end{aligned}$$

Discretize using finite elements (interlab), run with Matlab implementation of penalty method with:

- $n = 2050$, $m = 961$
- $\alpha = 10^{-4}$, $\sigma = 10^{-2}$
- TRON (Newton-CG) applied on $\phi_\sigma(u, z)$
- LNLQ for augmented system solves, rel. error tol $= 10^{-3}$

Inverse Poisson Problem

iter	merit	objective	opt Error	du Feas	nln Feas	CGits	stat
0	2.997845e+00	1.837255e+00	7.985e-03	7.123e-03	2.000e+00		
1	2.964378e+00	1.820891e+00	7.976e-03	7.087e-03	1.975e+00	1	
2	2.834288e+00	1.755744e+00	7.934e-03	6.938e-03	1.878e+00	1	
3	2.372450e+00	1.498781e+00	7.645e-03	6.225e-03	1.492e+00	1	
4	1.111925e+00	6.183138e-01	6.570e-03	7.431e-03	1.025e+00	1	
5	1.381301e-01	3.408713e-02	3.827e-03	2.515e-03	4.348e-01	2	
6	4.033638e-01	1.108718e-01	4.538e-03	3.096e-03	1.180e+00	2	rej
7	6.599361e-02	1.687954e-02	2.433e-03	2.230e-03	2.004e-01	2	
8	3.608916e-03	9.273160e-04	7.735e-04	4.657e-04	7.073e-02	2	
9	2.334628e-04	2.316148e-04	3.226e-05	1.783e-05	6.444e-03	1	
10	2.306002e-04	2.305667e-04	1.013e-06	8.145e-07	2.053e-04	1	
11	2.081017e-04	1.997098e-04	3.682e-05	2.142e-05	3.505e-03	6	
12	2.052722e-04	2.054533e-04	1.543e-06	8.706e-07	3.129e-04	1	
13	1.905605e-04	1.874832e-04	2.050e-05	1.217e-05	8.842e-04	10	
14	1.898997e-04	1.899435e-04	5.290e-07	3.753e-07	7.993e-05	1	
15	1.854960e-04	1.842241e-04	1.369e-05	8.012e-06	6.028e-04	16	
16	1.853261e-04	1.853453e-04	6.334e-07	3.400e-07	4.926e-05	1	
17	1.848716e-04	1.847335e-04	2.251e-06	1.229e-06	1.597e-04	24	
18	1.848673e-04	1.848701e-04	1.755e-07	9.599e-08	1.207e-05	1	
19	1.848203e-04	1.848129e-04	1.052e-07	5.783e-08	6.837e-06	32	
20	1.848202e-04	1.848205e-04	6.376e-09	3.621e-09	4.582e-07	1	

Inverse Poisson Problem

	Iterations	# Hv	# Jprod	# Adj. Jprod
$\epsilon = 10^{-2}$	22	878	3448	3672
$\epsilon = 10^{-4}$	21	896	4251	4459
$\epsilon = 10^{-6}$	20	744	4651	4928
$\epsilon = 10^{-8}$	20	746	5611	5887
$\epsilon = 10^{-10}$	20	746	6595	6871

ROL_example_PDE-OPT_navier-stokes_example_01

Fletcher solver : Trust_Region

iter	merit	fval	gpnorm	gLnorm	cnorm	snorm	flag	iterCG
0	3.1e-01	3.1e-01	4.1e-02	4.1e-02	4.6e-12			
1	3.1e-01	3.0e-01	4.1e-02	8.9e+00	2.5e-04	6.8e-01	2	48
2	3.1e-01	3.0e-01	4.1e-02	4.1e-02	4.6e-12	1.7e-01	0	2
3	3.1e-01	3.1e-01	9.2e-02	6.1e-02	1.4e-06	4.3e-02	0	1
4	3.0e-01	3.0e-01	5.3e-01	2.8e-01	8.6e-04	1.0e-01	0	4
5	3.0e-01	3.0e-01	4.4e-01	2.2e-01	5.1e-05	1.0e-01	0	7
6	3.0e-01	3.0e-01	9.8e-01	4.9e-01	1.2e-05	1.6e-01	0	42
7	3.0e-01	3.0e-01	1.1e-01	5.8e-02	4.1e-06	6.0e-02	0	31
8	3.0e-01	2.9e-01	4.2e-01	2.1e-01	5.1e-06	1.1e-01	0	39
9	2.9e-01	2.9e-01	8.2e-03	4.2e-03	7.3e-07	1.7e-02	0	25
10	2.9e-01	2.9e-01	6.6e-03	3.3e-03	8.2e-08	1.4e-02	0	27
11	2.9e-01	2.9e-01	5.3e-05	4.9e-05	5.2e-08	1.0e-04	0	7
12	2.9e-01	2.9e-01	1.7e-06	8.9e-07	1.9e-10	2.4e-04	0	44

Conclusions and Future Work

Simpler approach to nonlinear programming!

- Initial version implemented into ROL in only couple days!
- Shows promise when augmented systems efficiently solvable (e.g. PDE-constrained optimization)
- Current implementation Matlab, ROL implementation coming along

Conclusions and Future Work

Simpler approach to nonlinear programming!

- Initial version implemented into ROL in only couple days!
- Shows promise when augmented systems efficiently solvable (e.g. PDE-constrained optimization)
- Current implementation Matlab, ROL implementation coming along

Many practical matters to be resolved:

- Practical tolerance rules for inexact solves
- Good heuristics for updating penalty parameter
- Stability of inequality constrained problems near boundary
- Continue implementation in Matlab and ROL
- Test on more PDE-constrained optimization problems!