

Ipopt Tutorial

Andreas Wächter

IBM T.J. Watson Research Center

`andreasw@watson.ibm.com`

DIMACS Workshop on COIN-OR

DIMACS Center, Rutgers University

July 17, 2006

Outline

- Installation
- Using Ipopt from AMPL
- The algorithm behind Ipopt
- What are those 100 Ipopt options?
- Things to avoid when modeling NLPs
- Using Ipopt from your own code
- Coding example
- Open discussion

Where to get information

- Ipop home page

<https://projects.coin-or.org/Ipop>

- Wiki-based (contribution, changes, corrections are welcome!!!)
- Bug ticket system (click on “View Tickets”)

- Online documentation

<http://www.coin-or.org/Ipop/documentation/>

- Mailing list

<http://list.coin-or.org/mailman/listinfo/coin-ipopt>

- Main developers

- Andreas Wächter (project manager)
- Carl Laird

Downloading the code

- Obtaining the Ipopt code with subversion

```
$ svn co https://projects.coin-or.org/svn/Ipopt/trunk Coin-Ipopt  
$ cd Coin-Ipopt
```

- Obtaining third-party code from netlib

```
$ cd ThirdParty/ASL
```

```
$ ./get.ASL
```

(Ampl Solver Library)

```
$ cd ../Blas
```

```
$ ./get.Blas
```

(Basic Linear Algebra Subroutines)

```
$ cd ../Lapack
```

```
$ ./get.Lapack
```

(Linear Algebra PACKage)

```
$ cd ../..
```

- Obtain linear solver (MA27 and MC19) from Harwell-Archive
 - read Ipopt Download documentation (“Download HSL routines”)

Configuration and Compilation

- Preferred way: “VPATH Install” (objects separate from source)

```
$ mkdir build
```

```
$ cd build
```

This allows you easily to start over if necessary (delete `build` directory).

- Run the configuration script (here very basic version)

```
$ ../configure
```

This performs a number of tests (e.g., compiler choice and options), and creates directories with Makefiles.

Look for “`Main Ipopt configuration successful.`”

- Compile the code

```
$ make
```

- Test the compiled code

```
$ make test
```

- Install the executable, libraries, and header files

```
$ make install
```

into the `bin/`, `lib/`, and `include/` subdirectories

Advanced Configuration

- Choosing different compilers

```
$ ./configure [...] CXX=icpc CC=icc F77=ifc
```

- Choosing different compiler options

```
$ ./configure [...] CXXFLAGS="-O -pg" [CFLAGS=... FFLAGS=...]
```

- Compiling static instead of shared libraries

```
$ ./configure [...] --disable-shared
```

- Using different BLAS library (similarly for LAPACK)

```
$ ./configure [...] --with-blas="-L$HOME/lib -lf77blas -latlas"
```

- Using a different linear solver (e.g., Pardiso or WSMP)

```
$ ./configure [...] --with-pardiso="$HOME/lib/libwsmp_P4.a"
```

- Speeding up using cache for tests with flag **-c**

- IMPORTANT: Delete `config.cache` file before rerunning `configure`

- More information:

- Section "Detailed Installation Information" in Ipopt documentation
- <https://projects.coin-or.org/BuildTools/wiki/user-configure>

What to do if configuration fails?

- Look at output of the `configure` script
- For more details, look into the `config.log` file in directory where configuration failed:
 - Look for latest “`configuring in ...`” in `configure` output, e.g.
`config.status: executing depfiles commands`
`configure: Configuration of ThirdPartyASL successful`
`configure: configuring in Ipopt`
`configure: running /bin/sh '/home/andreasw/COI ...`
This tells you subdirectory name (e.g., `Ipopt`)
 - Open `config.log` file in that directory
 - Go to the bottom, and go back up until you see
`## ----- ##`
`## Cache variables. ##`
`## ----- ##`
 - Just before could be useful output corresponding to the error
- If you can't fix the problem, submit ticket (attach this `config.log` file!)

General NLP Problem Formulation

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & g_L \leq g(x) \leq g_U \\ & x_L \leq x \leq x_U \end{array}$$

x

$$f(x) : \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$g(x) : \mathbb{R}^n \longrightarrow \mathbb{R}^m$$

$$g_L \in (\mathbb{R} \cup \{-\infty\})^m$$

$$g_U \in (\mathbb{R} \cup \{\infty\})^m$$

$$x_L \in (\mathbb{R} \cup \{-\infty\})^n$$

$$x_U \in (\mathbb{R} \cup \{\infty\})^n$$

Continuous variables

Objective function

Constraints

Constraint bounds

Variable bounds

- Equality constraints with $g_L^{(i)} = g_U^{(i)}$
- Goal: Numerical method for finding *local* solution x_*
- Local solution x_* : Exists neighborhood U of x_* so that

$$\forall x \in U : \quad x \text{ feasible} \implies f(x) \geq f(x_*)$$

- We say, the problem is *convex*, if ...

Using Ipopt from AMPL

- You need:

- The AMPL interpreter `ampl`.

The student version is free; size limit 300 constraints/variables, see

<http://www.netlib.org/ampl/student/>

- The Ipopt AMPL solver executable `ipopt`

It is in the `bin/` subdirectory after `make install`.

- Make sure that both `ampl` and `ipopt` are in your path.

For example, copy both executables into `$HOME/bin` and set `PATH` to `$HOME/bin : $PATH` in your shell's startup script.

Basic AMPL commands

- Start AMPL (just type “`ampl`”)
- Select solver: `option solver ipopt;`
- Set Ipopt options: `option ipopt_options 'mu_strategy=adaptive ...';`
- Load an AMPL model: `model hs100.mod;`
- Solve the model: `solve;`
- Enjoy the Ipopt output *tic, tac, tic, tac...*
- Look at solution: `display x;`
- Before loading new model: `reset;`

Some examples can be downloaded here:

- Bob Vanderbei's AMPL model collection (incl. CUTE):
<http://www.sor.princeton.edu/~rvdb/ampl/nlmodels/>
- COPS problems
<http://www-unix.mcs.anl.gov/~more/cops/>

Problem Formulation With Slacks

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_L \leq g(x) \leq g_U \\ & x_L \leq x \leq x_U \end{aligned}$$

$$\begin{aligned} \mathcal{E} &= \{i : g_L^{(i)} = g_U^{(i)}\} \\ \mathcal{I} &= \{i : g_L^{(i)} < g_U^{(i)}\} \end{aligned} \longrightarrow$$

$$\begin{aligned} \min_{x, s} \quad & f(x) \\ \text{s.t.} \quad & g^{\mathcal{E}}(x) - g_L^{\mathcal{E}} = 0 \\ & g^{\mathcal{I}}(x) - s = 0 \\ & g_L^{\mathcal{I}} \leq s \leq g_U^{\mathcal{I}} \\ & x_L \leq x \leq x_U \end{aligned}$$

Simplified formulation for presentation of algorithm:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

Basics: Optimality conditions

- Try to find point that satisfies first-order optimality conditions:

$$\nabla f(x) + \nabla c(x)y - z = 0$$

$$c(x) = 0$$

$$XZe = 0$$

$$x, z \geq 0$$

$$e = (1, \dots, 1)^T$$

$$X = \text{diag}(x)$$

$$Z = \text{diag}(z)$$

- Multipliers:
 - y for equality constraints
 - z for bound constraints
- If original problem convex, then every such point is global solution
- Otherwise, also maxima and saddle points might satisfy those conditions

Assumptions

- Functions $f(x)$, $c(x)$ are sufficiently smooth:
Theoretically, C^1 for global convergence, C^2 for fast local convergence.
- The algorithm requires first derivatives of all functions, and if possible, second derivatives
- In theory, need Linear-Independence-Constraint-Qualification (LICQ):
The gradients of active constraints

$$\nabla c^{(i)}(x_*) \text{ for } i = 1, \dots, m \quad \text{and} \quad e_i \text{ for } x_*^{(i)} = 0$$

are linearly independent at solution x_* .

- For fast local convergence, need strong second-order optimality conditions:
 - Hessian of Lagrangian is positive definite in null space of active constraint gradients
 - Strict complementarity, i.e., $x_*^{(i)} + z_*^{(i)} > 0$ for $i = 1, \dots, n$

Barrier Method

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) \\ s.t. & c(x) = 0 \\ & x \geq 0 \end{array}$$

Barrier Method

$$\begin{array}{ll}\min_{x \in \mathbb{R}^n} & f(x) \\ s.t. & c(x) = 0 \\ & x \geq 0\end{array}$$



$$\begin{array}{ll}\min_{x \in \mathbb{R}^n} & f(x) - \mu \sum_{i=1}^n \ln(x^{(i)}) \\ s.t. & c(x) = 0\end{array}$$

Barrier Parameter: $\mu > 0$

Idea: $x_*(\mu) \rightarrow x_*$ as $\mu \rightarrow 0$.

Barrier Method

$$\begin{array}{ll}\min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & c(x) = 0 \\ & x \geq 0\end{array}$$



$$\begin{array}{ll}\min_{x \in \mathbb{R}^n} & f(x) - \mu \sum_{i=1}^n \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0\end{array}$$

Barrier Parameter: $\mu > 0$

Idea: $x_*(\mu) \rightarrow x_*$ as $\mu \rightarrow 0$.

Outer Algorithm

(Fiacco, McCormick (1968))

1. Given initial $x_0 > 0$, $\mu_0 > 0$. Set $l \leftarrow 0$.
2. Compute (approximate) solution x_{l+1} for BP(μ_l) with error tolerance $\epsilon(\mu_l)$.
3. Decrease barrier parameter μ_l (superlinearly) to get μ_{l+1} .
4. Increase $l \leftarrow l + 1$; go to 2.

Solution of the Barrier Problem

Barrier Problem (fixed μ)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & \varphi_\mu(x) := f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0 \end{array}$$

Solution of the Barrier Problem

Barrier Problem (fixed μ)

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & \varphi_\mu(x) := f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0 \end{array}$$

Optimality Conditions

$$\begin{array}{rcl} \nabla \varphi_\mu(x) + \nabla c(x)y & = & 0 \\ c(x) & = & 0 \\ (x & > & 0) \end{array}$$

Solution of the Barrier Problem

Barrier Problem (fixed μ)

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \varphi_\mu(x) := f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

Optimality Conditions

$$\begin{aligned} \nabla \varphi_\mu(x) + \nabla c(x)y &= 0 \\ c(x) &= 0 \\ (x &> 0) \end{aligned}$$

Apply Newton's Method

$$\begin{bmatrix} W_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k)y_k \\ c(x_k) \end{pmatrix}$$

Here:

- $W_k = \nabla_{xx}^2 \mathcal{L}_\mu(x_k, y_k)$
- $\mathcal{L}_\mu(x, y) = \varphi_\mu(x) + c(x)^T y$

Solution of the Barrier Problem

Barrier Problem (fixed μ)

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \varphi_\mu(x) := f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

Optimality Conditions

$$\begin{aligned} \nabla \varphi_\mu(x) + \nabla c(x)y &= 0 \\ c(x) &= 0 \\ (x > 0) \end{aligned}$$

Apply Newton's Method

$$\begin{bmatrix} W_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k)y_k \\ c(x_k) \end{pmatrix}$$

Here:

- $W_k = \nabla_{xx}^2 \mathcal{L}_\mu(x_k, y_k)$
- $\mathcal{L}_\mu(x, y) = \varphi_\mu(x) + c(x)^T y$

$$\begin{aligned} \nabla \varphi_\mu(x) &= \nabla f(x) - \mu X^{-1} e \\ \nabla^2 \varphi_\mu(x) &= \nabla^2 f(x) + \mu X^{-2} \\ X &:= \text{diag}(x) \quad e := (1, \dots, 1)^T \end{aligned}$$

Primal-Dual Approach

Primal

$$\begin{aligned}\nabla f(x) - \mu X^{-1}e + \nabla c(x)y &= 0 \\ c(x) &= 0 \\ (x &> 0)\end{aligned}$$

Primal-Dual Approach

Primal

$$\begin{aligned}\nabla f(x) - \mu X^{-1}e + \nabla c(x)y &= 0 \\ c(x) &= 0 \\ (x > 0)\end{aligned}$$

$$z = \mu X^{-1}e \longrightarrow$$

Primal-Dual

$$\begin{aligned}\nabla f(x) + \nabla c(x)y - z &= 0 \\ c(x) &= 0 \\ XZe - \mu e &= 0 \\ (x, z > 0)\end{aligned}$$

Primal-Dual Approach

Primal

$$\begin{aligned}\nabla f(x) - \mu X^{-1}e + \nabla c(x)y &= 0 \\ c(x) &= 0 \\ (x > 0)\end{aligned}$$

$$z = \mu X^{-1}e \xrightarrow{\quad}$$

Primal-Dual

$$\begin{aligned}\nabla f(x) + \nabla c(x)y - z &= 0 \\ c(x) &= 0 \\ XZe - \mu e &= 0 \\ (x, z > 0)\end{aligned}$$

Apply Newton's Method

$$\begin{bmatrix} W_k & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta y_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k)y_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu e \end{pmatrix}$$

$$\text{Now: } W_k = \nabla_{xx}^2 f(x_k) + \sum y_k^{(i)} \nabla_{xx}^2 c^{(i)}(x_k)$$

Line Search

Need to find $\alpha_k^x, \alpha_k^y, \alpha_k^z \in (0, 1]$ to obtain new iterates

$$x_{k+1} = x_k + \alpha_k^x \Delta x_k$$

$$y_{k+1} = y_k + \alpha_k^y \Delta y_k$$

$$z_{k+1} = z_k + \alpha_k^z \Delta z_k$$

Line Search

Need to find $\alpha_k^x, \alpha_k^y, \alpha_k^z \in (0, 1]$ to obtain new iterates

$$x_{k+1} = x_k + \alpha_k^x \Delta x_k$$

$$y_{k+1} = y_k + \alpha_k^y \Delta y_k$$

$$z_{k+1} = z_k + \alpha_k^z \Delta z_k$$

1. Keep x_k and z_k positive (“fraction-to-the-boundary rule”):
Determine largest $\alpha_k^{x,\tau}, \alpha_k^{z,\tau} \in (0, 1]$ such that $(\tau \in (0, 1), \text{ close to } 1)$

$$x_k + \alpha_k^{x,\tau} \Delta x_k \geq (1 - \tau)x_k > 0$$

$$z_k + \alpha_k^{z,\tau} \Delta z_k \geq (1 - \tau)z_k > 0$$

Line Search

Need to find $\alpha_k^x, \alpha_k^y, \alpha_k^z \in (0, 1]$ to obtain new iterates

$$x_{k+1} = x_k + \alpha_k^x \Delta x_k$$

$$y_{k+1} = y_k + \alpha_k^y \Delta y_k$$

$$z_{k+1} = z_k + \alpha_k^z \Delta z_k$$

1. Keep x_k and z_k positive (“fraction-to-the-boundary rule”):
Determine largest $\alpha_k^{x,\tau}, \alpha_k^{z,\tau} \in (0, 1]$ such that $(\tau \in (0, 1), \text{ close to } 1)$

$$x_k + \alpha_k^{x,\tau} \Delta x_k \geq (1 - \tau)x_k > 0$$

$$z_k + \alpha_k^{z,\tau} \Delta z_k \geq (1 - \tau)z_k > 0$$

2. Backtracking line search $\alpha_{k,l}^x = 2^{-l} \alpha_k^{x,\tau}$ to ensure global convergence

Line Search

Need to find $\alpha_k^x, \alpha_k^y, \alpha_k^z \in (0, 1]$ to obtain new iterates

$$x_{k+1} = x_k + \alpha_k^x \Delta x_k$$

$$y_{k+1} = y_k + \alpha_k^y \Delta y_k$$

$$z_{k+1} = z_k + \alpha_k^z \Delta z_k$$

$\alpha_{k,l}^x$ from line search

see `alpha_for_y`

is $\alpha_k^{z,\tau}$

1. Keep x_k and z_k positive (“fraction-to-the-boundary rule”):
Determine largest $\alpha_k^{x,\tau}, \alpha_k^{z,\tau} \in (0, 1]$ such that $(\tau \in (0, 1), \text{ close to } 1)$

$$x_k + \alpha_k^{x,\tau} \Delta x_k \geq (1 - \tau)x_k > 0$$

$$z_k + \alpha_k^{z,\tau} \Delta z_k \geq (1 - \tau)z_k > 0$$

2. Backtracking line search $\alpha_{k,l}^x = 2^{-l} \alpha_k^{x,\tau}$ to ensure global convergence

***Ipop** Options*

- List available options:
 - “Options Reference” section in Ipop documentation
 - Run AMPL solver executable
`$./ipop=-`
 - Use option `print_options_documentation = yes`
Prints list of all Ipop options with explanation
- Setting options:
 - From AMPL (`option ipopt_options 'op1=val1 ...';`)
 - From NLP program code
 - Using options file (“`ipop.opt`”); has priority
- Output options:
 - “`print_level`”: determines amount of screen output
 - “`output_file`”: name of output file (default: none)
 - “`file_print_level`”: amount of output into file
 - “`print_info_string`”: “`yes`” shows cryptic additional information

Successful Termination

- Components of the optimality conditions:

$$E_d = \|\nabla f(x) + \nabla c(x)y - z\|_\infty$$

Dual infeasibility

$$E_p = \|c(x)\|_\infty$$

Primal infeasibility

$$E_c^\mu = \|XZe - \mu e\|_\infty$$

Complementarity error

- Overall optimality error:

$$E_{nlp}^\mu = \max \left\{ \frac{E_d}{s_d}, E_p, \frac{E_c^\mu}{s_c} \right\}$$

- Error scaling factors: $s_d = \max \left\{ 1, \frac{\|y\|_1 + \|z\|_1}{(n+m) \mathbf{s_max}} \right\}$ $s_c = \max \left\{ 1, \frac{\|z\|_1}{n \mathbf{s_max}} \right\}$

- Successful termination if

$$E_{nlp}^0 \leq \mathbf{tol} \quad \text{and} \quad \begin{cases} E_d \leq \mathbf{dual_inf_tol} & \text{and} \\ E_p \leq \mathbf{constr_viol_tol} & \text{and} \\ E_c^0 \leq \mathbf{compl_inf_tol} \end{cases}$$

Other Termination Criteria

- Maximum number of iterations exceeded

$$k \geq \text{max_iter}$$

- “Acceptable tolerance” criterium

$$E_{nlp}^0 \leq \text{acceptable_tol}$$

and

$$\left\{ \begin{array}{ll} E_d \leq \text{acceptable_dual_inf_tol} & \text{and} \\ E_p \leq \text{acceptable_constr_viol_tol} & \text{and} \\ E_c^0 \leq \text{acceptable_compl_inf_tol} \end{array} \right.$$

satisfied in `acceptable_iter` consecutive iterations

- Iterates seem to diverge (unbounded problem?)

$$\|x_k\|_{\infty} \geq \text{diverging_iterates_tol}$$

Monotone Barrier Parameter Update

- Option `mu_strategy = monotone` (Fiacco-McCormick; default)
- Sequential solution of barrier problems:

1. Initialize $\mu_0 \leftarrow \text{mu_init}$ in first iteration
2. At beginning of each iteration, check if $E_{nlp}^{\mu_k} \leq \text{barrier_tol_factor} \cdot \mu_k$, then

$$\mu_k \leftarrow \min \left\{ \text{mu_max}, \max \left\{ \text{mu_min}, \min \left\{ \kappa_\mu \mu_k, \mu_k^{\theta_\mu} \right\} \right\} \right\}, \text{ where}$$

$$\kappa_\mu = \text{mu_linear_decrease_factor}$$

$$\theta_\mu = \text{mu_superlinear_decrease_power}$$

Otherwise, keep μ_k

3. Compute search direction and step sizes, update iterates
4. Set $\mu_{k+1} \leftarrow \mu_k$ and go back to 2

Adaptive Barrier Parameter Update

- Option `mu_strategy = adaptive`
- In each iteration, compute a new value of μ_k , using a “ μ -oracle”
- Monitor overall progress towards solution (based on option `adaptive_mu_globalization`):
If iterates do not make good progress, switch to monotone mode, until enough progress was made
- Possible choices of `mu_oracle`:
 - `quality-function`: Choose value of μ so that step to the boundary gives best progress in a quality function (uses linear model of optimality conditions)
 - `probing`: Use Mehrotra’s probing heuristic
 - `loqo`: Use LOQO’s formula`quality-function` and `probing` require extra step computation
- Often: Less iteration than monotone strategy; more CPU time per iteration

Step Computation

- Computation of search direction

$$\begin{bmatrix} W_k & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & 0 & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k) \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu e \end{pmatrix}$$

Step Computation

- Computation of search direction (from symmetric system)

$$\begin{bmatrix} W_k + \Sigma_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \lambda_k \\ c(x_k) \end{pmatrix}$$

$$\Delta z_k = \mu X_k^{-1} e - z_k - \Sigma_k \Delta x_k \quad \Sigma_k = X_k^{-1} Z_k$$

Step Computation

- Computation of search direction (from symmetric system)

$$\begin{bmatrix} W_k + \Sigma_k + \delta_1 I & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \lambda_k \\ c(x_k) \end{pmatrix}$$

$$\Delta z_k = \mu X_k^{-1} e - z_k - \Sigma_k \Delta x_k \quad \Sigma_k = X_k^{-1} Z_k$$

- Need to guarantee descent properties:
 - Ensure $W_k + \Sigma_k + \delta_1 I$ is positive definite in null space of $\nabla c(x_k)^T$
 - Choose appropriate $\delta_1 \geq 0$ (see `*_hessian_perturbation`)

Step Computation

- Computation of search direction (from symmetric system)

$$\begin{bmatrix} W_k + \Sigma_k + \delta_1 I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \lambda_k \\ c(x_k) \end{pmatrix}$$

$$\Delta z_k = \mu X_k^{-1} e - z_k - \Sigma_k \Delta x_k \quad \Sigma_k = X_k^{-1} Z_k$$

- Need to guarantee descent properties:
 - Ensure $W_k + \Sigma_k + \delta_1 I$ is positive definite in null space of $\nabla c(x_k)^T$
 - Choose appropriate $\delta_1 \geq 0$ (see `*_hessian_perturbation`)
- Heuristic for dependent constraints ($\nabla c(x_k)$ rank deficient):
 - Choose small $\delta_2 > 0$ if matrix singular (see `jacobian_regularization_value`)

Step Computation

- Computation of search direction (from symmetric system)

$$\begin{bmatrix} W_k + \Sigma_k + \delta_1 I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \end{pmatrix} = - \begin{pmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k) \lambda_k \\ c(x_k) \end{pmatrix}$$

$$\Delta z_k = \mu X_k^{-1} e - z_k - \Sigma_k \Delta x_k \quad \Sigma_k = X_k^{-1} Z_k$$

- Need to guarantee descent properties:
 - Ensure $W_k + \Sigma_k + \delta_1 I$ is positive definite in null space of $\nabla c(x_k)^T$
 - Choose appropriate $\delta_1 \geq 0$ (see `*_hessian_perturbation`)
- Heuristic for dependent constraints ($\nabla c(x_k)$ rank deficient):
 - Choose small $\delta_2 > 0$ if matrix singular (see `jacobian_regularization_value`)
- Iterative refinement

Step Computation

- Computation of search direction (from symmetric system)

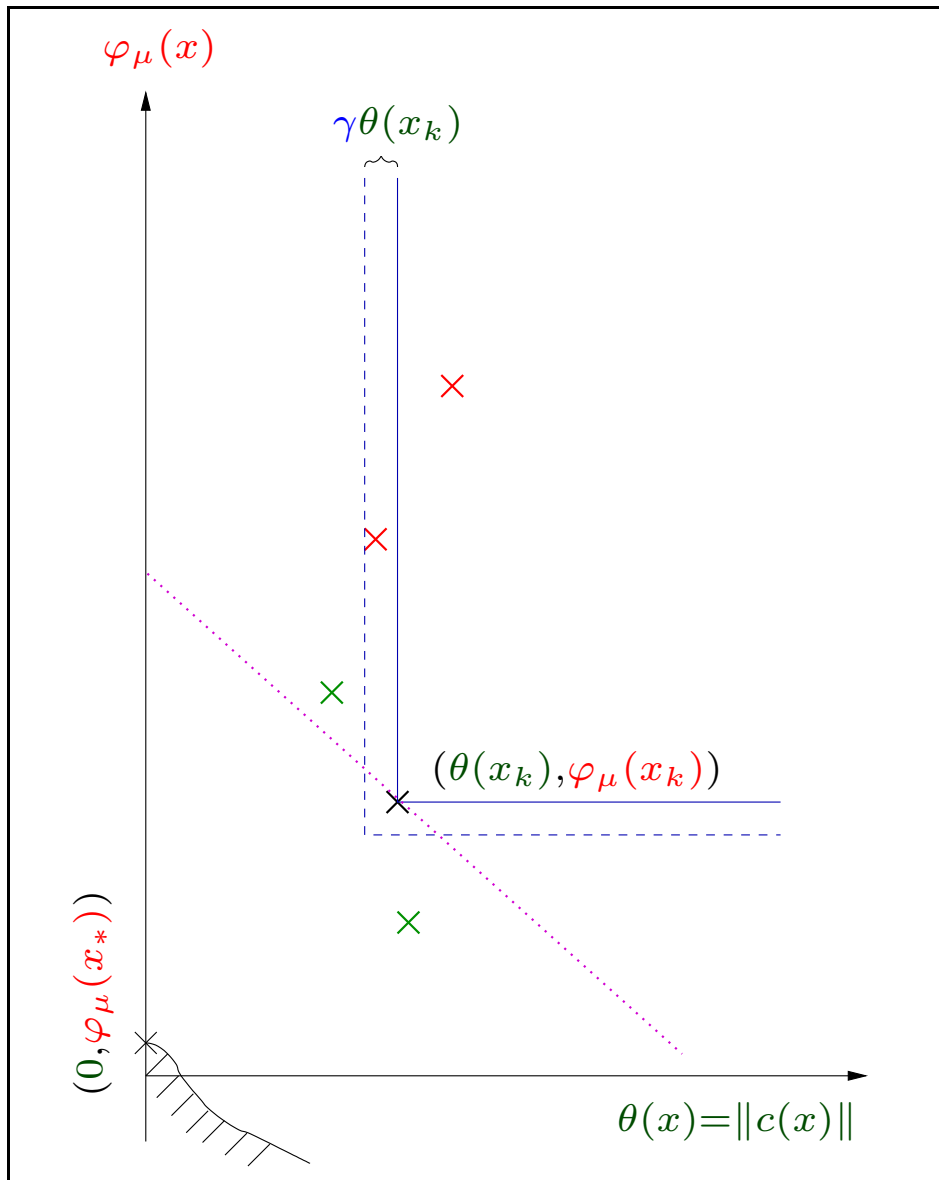
$$\begin{bmatrix} W_k + \delta_1 I & \nabla c(x_k) & -I \\ \nabla c(x_k)^T & -\delta_2 I & 0 \\ Z_k & 0 & X_k \end{bmatrix} \begin{pmatrix} \Delta x_k \\ \Delta \lambda_k \\ \Delta z_k \end{pmatrix} = - \begin{pmatrix} \nabla f(x_k) + \nabla c(x_k) \lambda_k - z_k \\ c(x_k) \\ X_k Z_k e - \mu e \end{pmatrix}$$

- Need to guarantee descent properties:
 - Ensure $W_k + \Sigma_k + \delta_1 I$ is positive definite in null space of $\nabla c(x_k)^T$
 - Choose appropriate $\delta_1 \geq 0$ (see `*_hessian_perturbation`)
- Heuristic for dependent constraints ($\nabla c(x_k)$ rank deficient):
 - Choose small $\delta_2 > 0$ if matrix singular (see `jacobian_regularization_value`)
- Iterative refinement on *full* system (see `min_refinement_steps`, `max_refinement_steps`)

Step Computation Options

- `linear_solver`: Linear solver to be used (availability depends on compilation)
Currently: MA27, MA57, Pardiso, WSMP, MUMPS
- `ma27_pivtol`, `ma27_pivtolmax`: Pivot tolerance for MA27
 - Larger: more exact computation of search direction
 - Smaller: faster computation time (less fill-in)
- `ma27_liw_init_factor`, `ma27_la_init_factor`, `ma27_meminc_factor`
Handle MA27's memory requirement; might help if you see:
`MA27BD returned iflag=-4 and requires more memory.`
Increase `liw` from 44960 to 449600 and `la` from 78400 to 794050 and factorize again.
- Scaling of the linear system (with MC19)
Usually gives better accuracy, but requires time
 - `linear_system_scaling`: “none” will prevent scaling
 - `linear_scaling_on_demand`: “yes” (if accur. bad), “no” (use always)

A Filter Line Search Method



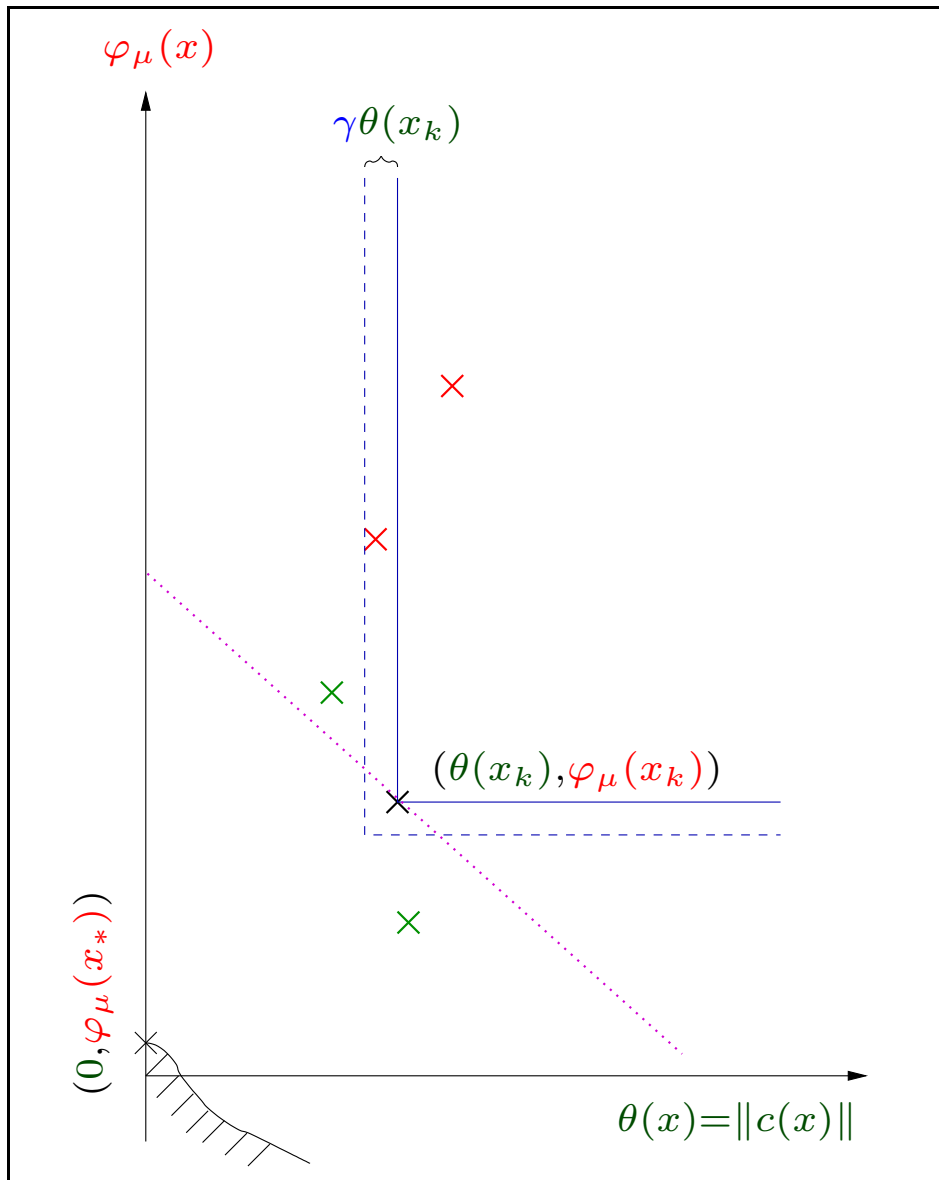
Idea: Bi-objective optimization
(Fletcher, Leyffer; 1998)

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

$$\min \theta(x)$$

$$\min \varphi_\mu(x)$$

A Filter Line Search Method



Idea: Bi-objective optimization
(Fletcher, Leyffer; 1998)

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

$$\min \theta(x)$$

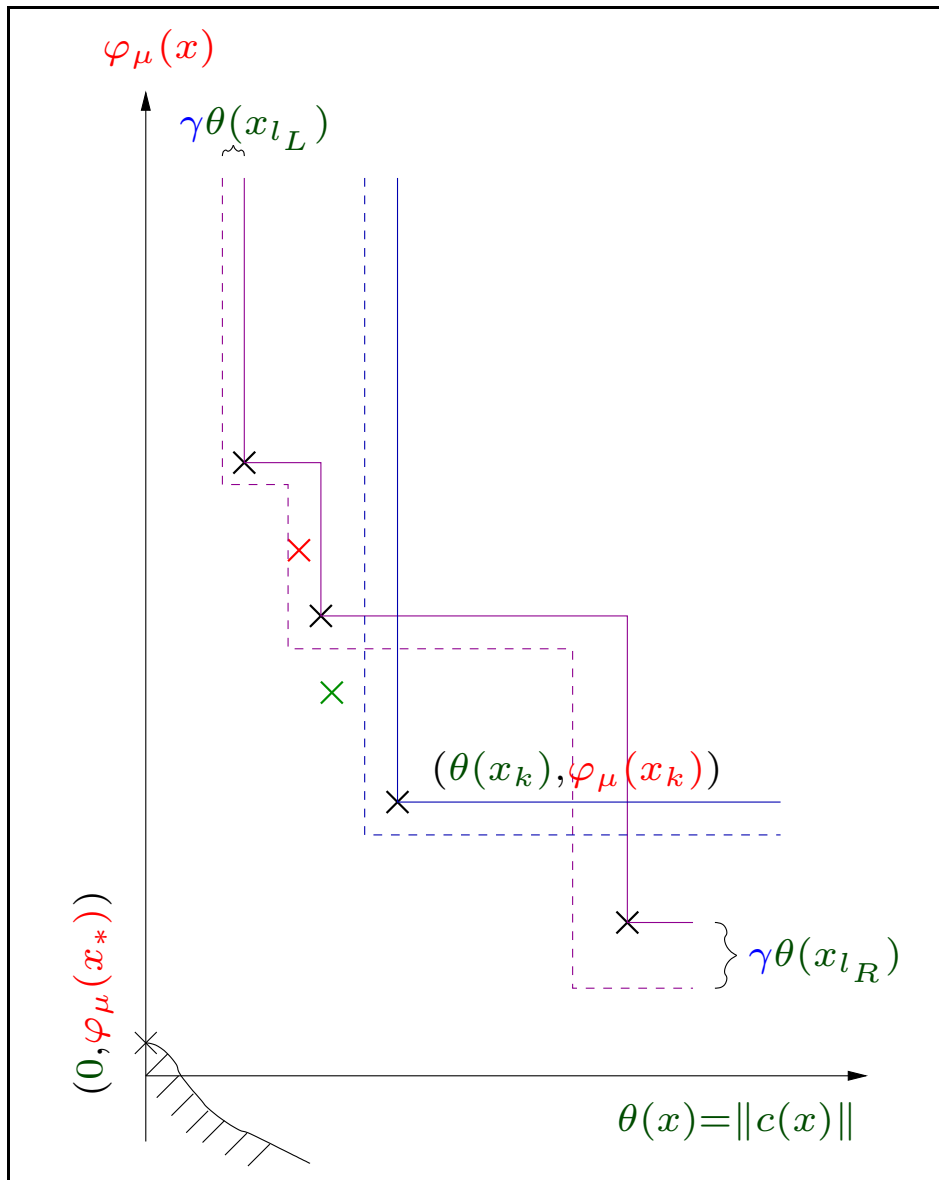
$$\min \varphi_\mu(x)$$

$$x_{\text{tr}} = x_k + \alpha \Delta x_k$$

Sufficient progress w.r.t. x_k :

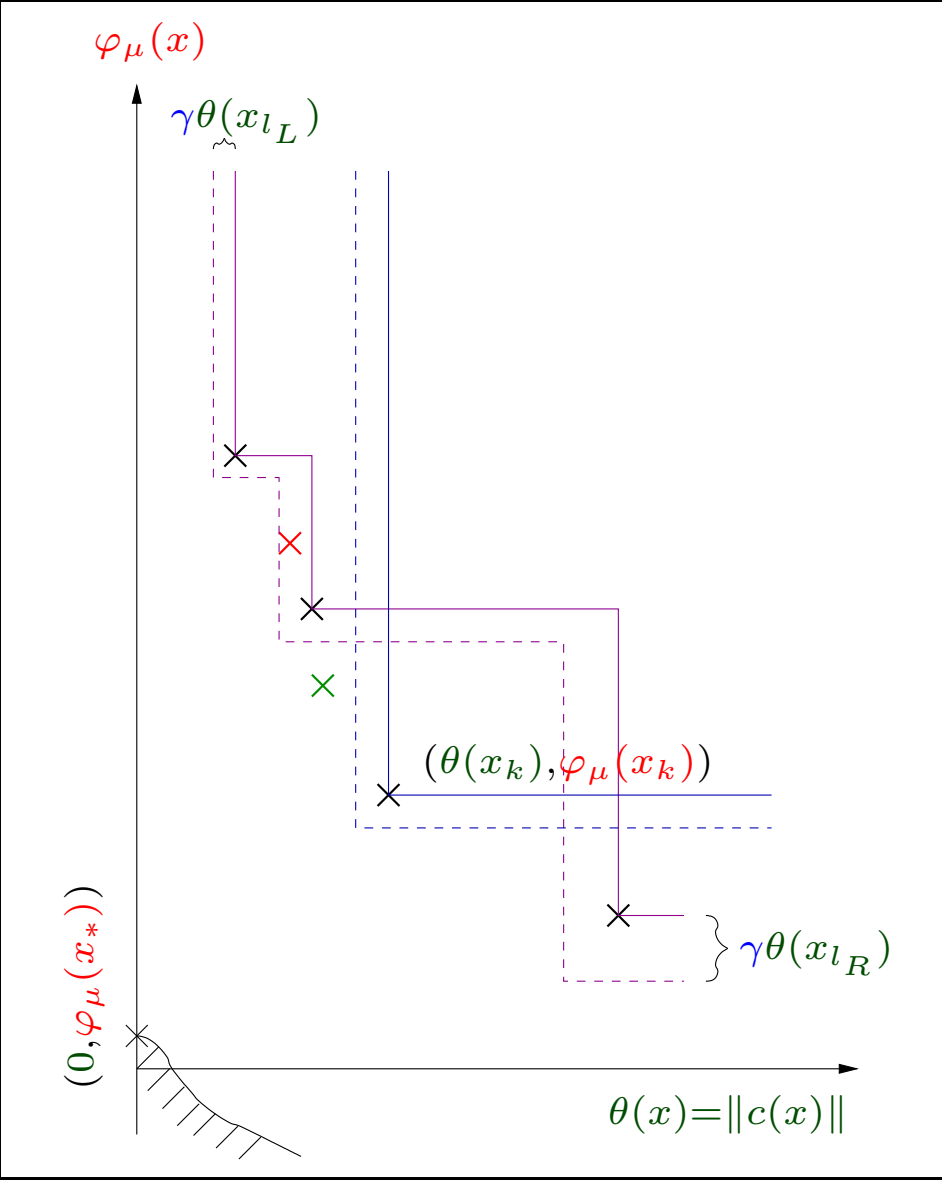
$$\begin{array}{ll} \varphi_\mu(x_{\text{tr}}) & \leq \varphi_\mu(x_k) - \gamma_\varphi \theta(x_k) \\ \theta(x_{\text{tr}}) & \leq \theta(x_k) - \gamma_\theta \theta(x_k) \end{array}$$

A Filter Line Search Method (Filter)



Need to avoid cycling

A Filter Line Search Method (Filter)



Need to avoid cycling



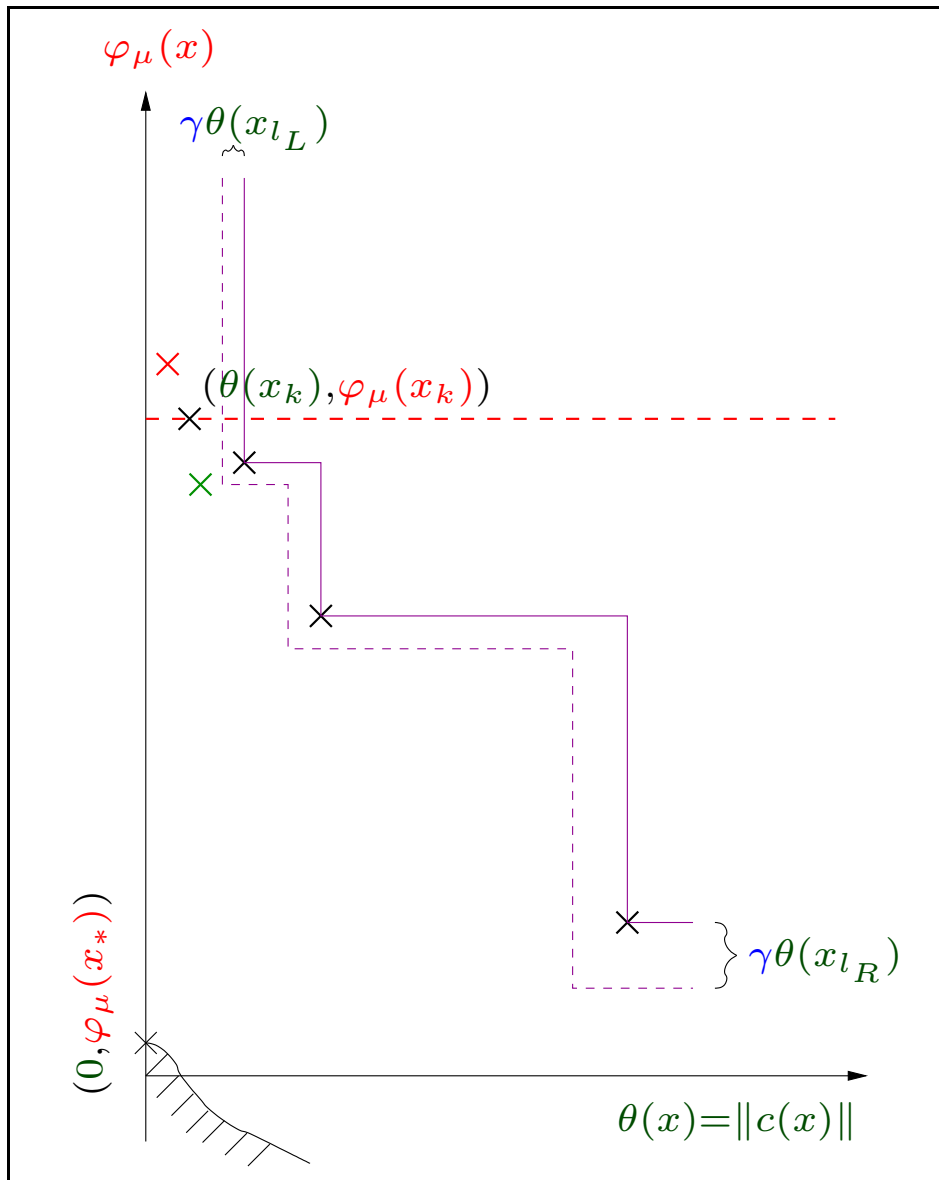
Store some previous
 $(\theta(x_l), \varphi_\mu(x_l))$ pairs in *filter* $\mathcal{F}_k$

Sufficient progress w.r.t. filter:

$$\begin{aligned}\varphi_\mu(\mathbf{x}_{\text{tr}}) &\leq \varphi_\mu(x_l) - \gamma_\varphi \theta(x_l) \\ \theta(\mathbf{x}_{\text{tr}}) &\leq \theta(x_l) - \gamma_\theta \theta(x_l)\end{aligned}$$

for $(\theta(x_l), \varphi_\mu(x_l)) \in \mathcal{F}_k$

A Filter Line Search Method (“ φ -type”)



If switching condition

$$-\alpha \nabla \varphi_\mu(x_k)^T \Delta x_k > \delta [\theta(x_k)]^{s_\theta}$$

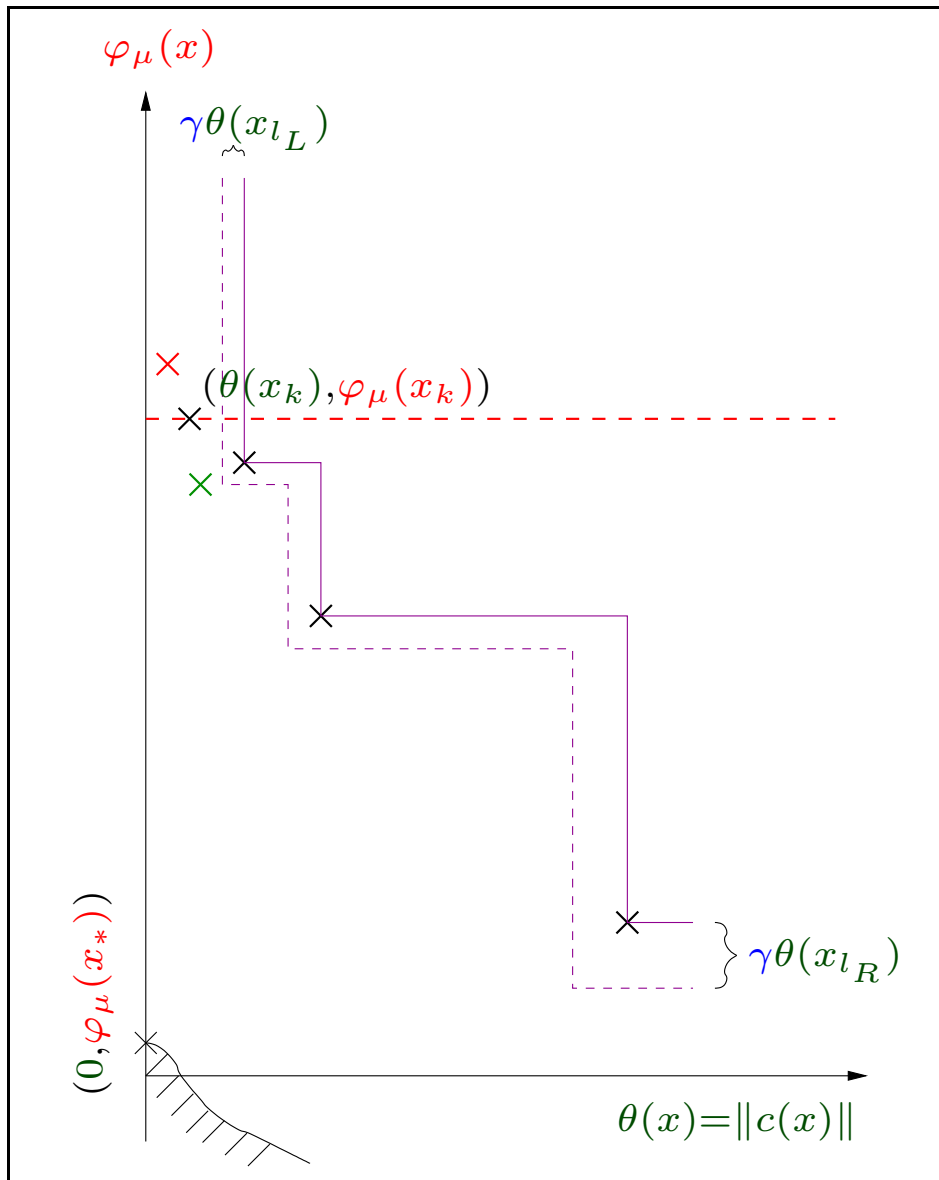
holds ($s_\theta > 1$):

$\Downarrow$

Armijo-condition on $\varphi_\mu(x)$:

$$\varphi_\mu(x_{\text{tr}}) \leq \varphi_\mu(x_k) + \alpha \eta \nabla \varphi_\mu(x_k)^T \Delta x_k$$

A Filter Line Search Method (“ φ -type”)



If switching condition

$$-\alpha \nabla \varphi_\mu(x_k)^T \Delta x_k > \delta [\theta(x_k)]^{s_\theta}$$

holds ($s_\theta > 1$):

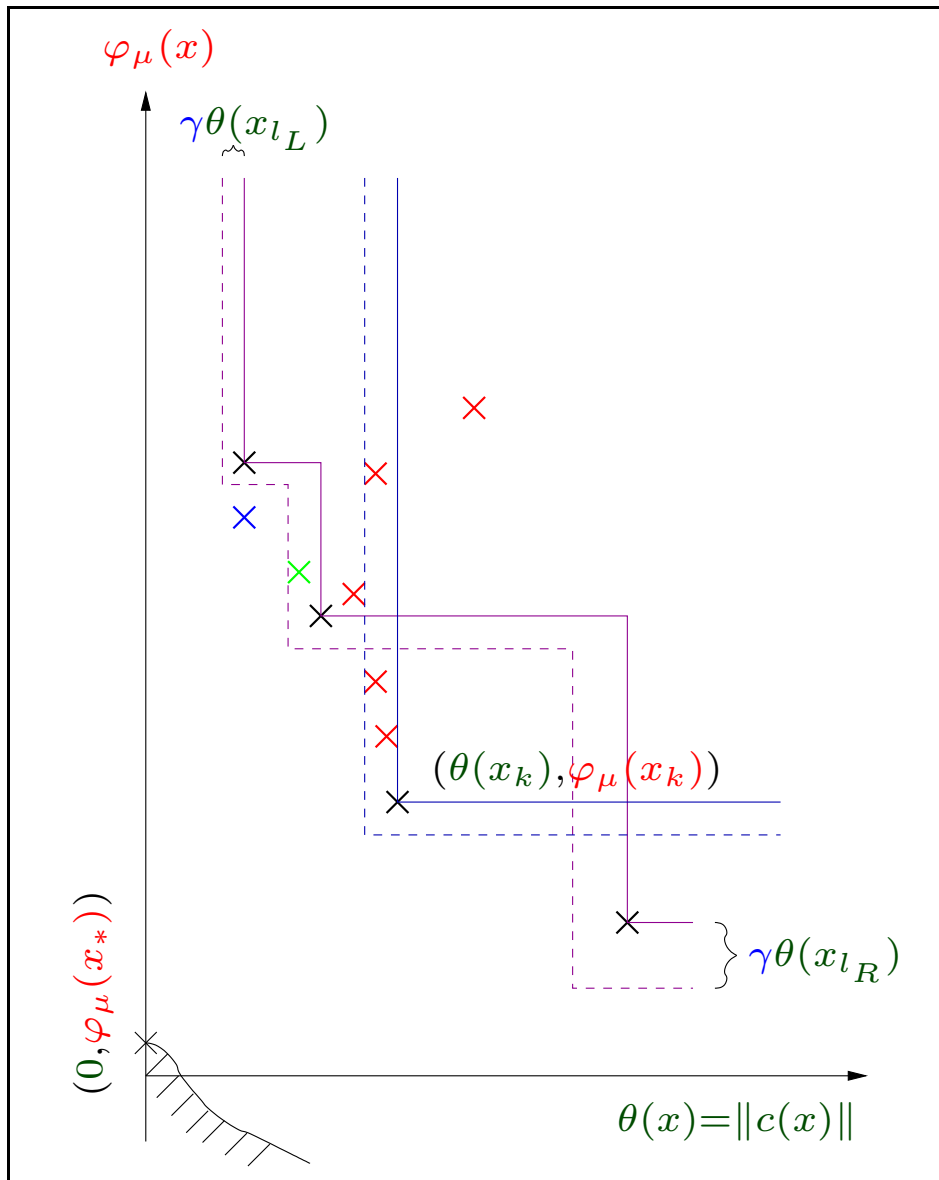
$\Downarrow$

Armijo-condition on $\varphi_\mu(x)$:

$$\varphi_\mu(x_{tr}) \leq \varphi_\mu(x_k) + \alpha \eta \nabla \varphi_\mu(x_k)^T \Delta x_k$$

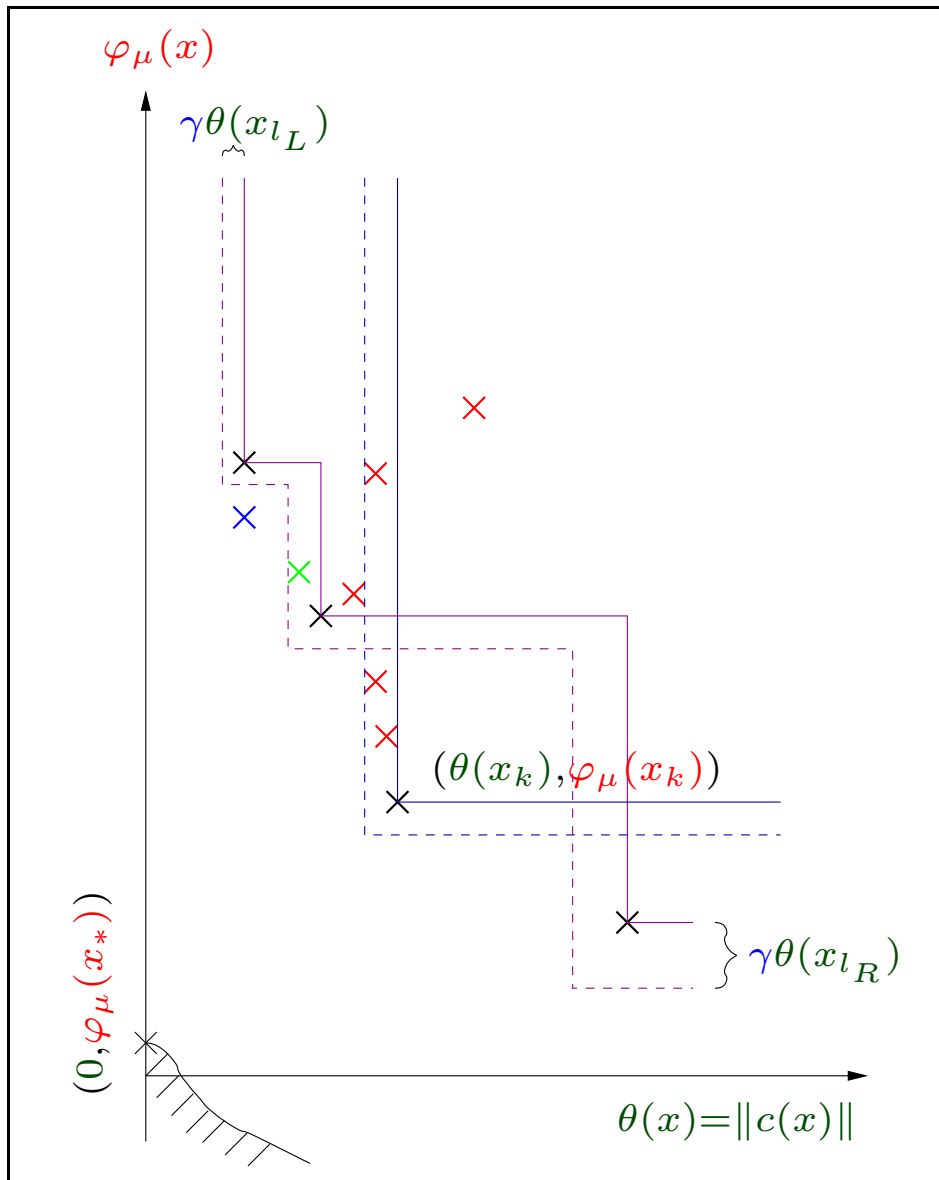
$\Rightarrow$ Don't augment $\mathcal{F}_k$ in that case

A Filter Line Search Method (Restoration)



If no admissible step size α_k can be found

A Filter Line Search Method (Restoration)



If no admissible step size α_k
can be found



Revert to
feasibility restoration phase:

Decrease $\theta(x)$ until

- found acceptable new
iterate $x_{k+1} := \tilde{x}_*^R > 0$, or
- converged to local mini-
mizer of constraint violation

Restoration Phase

$$\begin{array}{ll}\min & \|c(x)\| \\ s.t. & \\ & x \geq 0\end{array}$$

- Want to minimize constraint violation

Restoration Phase

$$\begin{array}{ll}\min & \|c(x)\|_1 + \eta \|x - \bar{x}_k\|_2^2 \\ s.t. & \\ & x \geq 0\end{array}$$

- Want to minimize constraint violation
- Control distance from “starting point” $\bar{x}_k$
 - Exact penalty formulation for “Find closest feasible point”:

$$\begin{array}{ll}\min & \|x - \bar{x}_k\|_2^2 \\ s.t. & c(x) = 0, \quad x \geq 0\end{array}$$

- Stabilizes Hessian non-singular (solution unique)

Restoration Phase

$$\begin{aligned} \min \quad & \sum \left(p^{(j)} + n^{(j)} \right) + \eta \|x - \bar{x}_k\|_2^2 \\ \text{s.t.} \quad & c(x) - p + n = 0 \\ & p, n, x \geq 0 \end{aligned}$$

- Want to minimize constraint violation
- Control distance from “starting point” $\bar{x}_k$
 - Exact penalty formulation for “Find closest feasible point”:

$$\begin{aligned} \min \quad & \|x - \bar{x}_k\|_2^2 \\ \text{s.t.} \quad & c(x) = 0, \quad x \geq 0 \end{aligned}$$

- Stabilizes Hessian non-singular (solution unique)
- Can be formulated smoothly
- Solve with interior point approach

Restoration Phase

$$\begin{aligned} \min \quad & \sum \left(p^{(j)} + n^{(j)} \right) + \eta \|x - \bar{x}_k\|_2^2 \\ & - \mu \sum \ln(x^{(i)}) - \mu \sum \ln(p^{(j)}) - \mu \sum \ln(n^{(j)}) \\ \text{s.t.} \quad & c(x) - p + n = 0 \end{aligned}$$

- Solve with interior point approach ($\eta = \sqrt{\mu} \rightarrow 0$)
 - Return $x_{k+1} = \tilde{x}_*^R > 0$
- Filter line search
- Step computation involves same matrix as in regular iteration
- Restoration phase simple:
 - Fix $x \implies$ Problem becomes separable
 - Solve analytically w.r.t. p and n

Restoration phase options

- Trigger for restoration depends on `alpha_min_frac`
- Restoration phase is finished if
 - constraint violation is reduced by factor `required_infeasibility_reduction`
 - restoration phase problem is converged; Ipopt terminates
⇒ Problem locally infeasible?
- Option `expect_infeasible_problem = yes`
 - triggers restoration phase earlier and demands more reduction in $\|c(x)\|$ (only first time).
- Algorithm for solving restoration phase is also Ipopt
 - Options can be set differently for restoration phase
 - Use “prefix” `resto.`, e.g.

`resto.mu_strategy = adaptive`

Second order corrections

- Maratos effect: Full step increases both $\varphi_\mu(x)$ and $\theta(x)$
 - (Can result in poor local convergence)
 - Here solve barrier problems only approximately

Second order corrections

- Maratos effect: Full step increases both $\varphi_\mu(x)$ and $\theta(x)$
 - (Can result in poor local convergence)
 - Here solve barrier problems only approximately
- Second order corrections

$$\begin{bmatrix} W_k + \Sigma_k + \delta_1 I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} \Delta x_k^{soc} \\ \Delta \lambda_k^{soc} \end{pmatrix} = - \begin{pmatrix} 0 \\ c(x_k + \alpha_k^{x,\tau} \Delta x_k) \end{pmatrix}$$

- additional Newton steps for the constraints (maximal `max_soc`)
- try $x_{tr} = x_k + \alpha_k^{\tau,soc} (\alpha_k^{x,\tau} \Delta x_k + \Delta x_k^{soc})$

Second order corrections

- Maratos effect: Full step increases both $\varphi_\mu(x)$ and $\theta(x)$
 - (Can result in poor local convergence)
 - Here solve barrier problems only approximately
- Second order corrections

$$\begin{bmatrix} W_k + \Sigma_k + \delta_1 I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} \Delta x_k^{soc} \\ \Delta \lambda_k^{soc} \end{pmatrix} = - \begin{pmatrix} 0 \\ c(x_k + \alpha_k^{x,\tau} \Delta x_k) \end{pmatrix}$$

- additional Newton steps for the constraints (maximal max_soc)
 - try $x_{tr} = x_k + \alpha_k^{\tau, soc} (\alpha_k^{x,\tau} \Delta x_k + \Delta x_k^{soc})$
- Helps to reduce the number of iterations
- “Anti-blocking” heuristics if repeated rejections in subsequent iterations
 - Reinitialize filter
 - Watchdog technique

Initialization

- User must provide x_0
(AMPL chooses zero for you, unless you set it!)
- Slack variables for inequality constraints $g^{\mathcal{I}}(x) - s = 0$ are set to $s_0 = g^{\mathcal{I}}(x_0)$.
- x and s variables are moved strictly inside bounds, based on
 - **bound_push**: Absolute distance from one bound
 - **bound_frac**: Relative distance between two bounds
- Bound multipliers z_0 are initialized to 1 (or **bound_mult_init_val**)
- Constraint multipliers y_0 are computed as least-square estimates for dual infeasibility

$$\|\nabla f(x_0) + \nabla c(x_0)y - z_0\|_2$$

If $\|y_0\| > \text{constr_mult_init_max}$, set $y_0 = 0$.

Scaling of the problem formulation

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & c(x) = 0 \\ & x \geq 0 \end{array}$$

- Idea: Problem is well-scaled if non-zero partial derivatives are typically of order 1
- Two ways to change problem scaling
 - Replace problem function $c^{(i)}(x)$ by $\tilde{c}^{(i)}(x) = s_c^{(i)} \cdot c^{(i)}(x)$ (similarly for $f(x)$)
 - Replace variable $x^{(i)}$ by $\tilde{x}^{(i)} = s_x^{(i)} \cdot x^{(i)}$
- Automatic scaling heuristic (`nlp_scaling_method`):
 - Scale each function $h(x)(= f(x), c^{(i)}(x))$ down, so that $\|h(x_0)\|_\infty \leq \text{nlp_scaling_max_gradient}$

Further Options

- `obj_scaling_factor`: Internal scaling factor for objective function
- `bound_relax_factor`: Bounds are relaxed slightly by this relative factor to create an interior
- `honor_original_bounds`: Even if bounds are relaxed internally, the returned solution will be in bounds.
- L-BFGS approximation of Lagrangian Hessian W_k
 - Active, if `hessian_approximation` is set to `limited_memory`
 - Can be used if second derivatives are not available
 - Usually less robust and slower
 - Can be useful if exact Hessian matrix is dense

Considerate modeling I

- Avoid nonlinearities if possible, e.g.

$$\prod_i x_i = c \iff \sum_i \log(x_i) = \log(c)$$

$$x/y = c \iff x = c \cdot y$$

- Try to formulate well scaled problems (sensitivities on the order of 1)
 - Multiply objective or constraint functions with constants
 - Variable transformation

$$\tilde{x} \leftarrow c \cdot x \quad \text{or} \quad \tilde{x} \leftarrow \phi(x)$$

- Try to have an interior, in particular, don't write

$$g(x) \leq 0 \quad \text{and} \quad g(x) \geq 0$$

- Skip unnecessary bounds

Considerate modeling II

- Try to have all functions be evaluable at all $x_L \leq x \leq x_U$, e.g.,

$$\log(z) \dots \text{ with } z = g(x), z \geq 0$$

instead of

$$\log(g(x)) \dots$$

if $g(x)$ can become negative

- Modeling binary or integer constraints as

$$x(x - 1) = 0$$

usually doesn't work.

- Try to avoid degenerate constraints

Writing an NLP as program

- Read the “Interfacing your NLP to IPOPT: A tutorial example” section in the Ipopt documentation.
 - What information is to be provided?
(Size, problem function values and derivatives, etc.)
 - How does Ipopt expect this information?
(e.g., sparse matrix format, C++ classes, `SmartPtr`’s...)
- Look at the code examples in `Coin-Ipopt/Ipopt/examples/*`
- Get example Makefile from `Coin-Ipopt/build/Ipopt/examples/*`
- Adapt examples (or start from scratch) for your problem
 - Set problem size, bounds, starting point etc.
 - Implement $f(x)$, $\nabla f(x)$, $g(x)$, $\nabla g(x)$
 - Set `derivative_test` to `first-order` and verify
(for small instance, if possible!)
 - When Ok, take care of $\nabla^2 \mathcal{L}$
(check with `derivative_test = second-order`)

Exercise example

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \sum_{i=1}^n (x_i - 1)^2 \\ \text{s.t.} \quad & (x_i^2 + 1.5x_i - a_i) \cos(x_{i+1}) - x_{i-1} = 0 \quad \text{for } i = 2, \dots, n-1 \\ & -1.5 \leq x_i \leq 0 \quad \text{for } i = 1, \dots, n \end{aligned}$$

- Starting point $(-0.5, \dots, -0.5)^T$
- Data $a_i = \frac{i}{n}$ (do not hardcode)
- See AMPL file `exercise_example.mod`
- URL:

Handling unbounded solution sets

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0 \end{array}$$

- What if original problem has unbounded set $\mathcal{S}$ of optimal solutions?

Handling unbounded solution sets

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0 \end{array}$$

- What if original problem has unbounded set $\mathcal{S}$ of optimal solutions?
- Then, $\varphi_\mu(\bar{x}_l) \rightarrow -\infty$ for some $\bar{x}_l \in \mathcal{S}$ with $\bar{x}_l \rightarrow \infty$

Handling unbounded solution sets

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) - \mu \sum \ln(x^{(i)}) \\ \text{s.t.} & c(x) = 0 \end{array}$$

- What if original problem has unbounded set $\mathcal{S}$ of optimal solutions?
- Then, $\varphi_\mu(\bar{x}_l) \rightarrow -\infty$ for some $\bar{x}_l \in \mathcal{S}$ with $\bar{x}_l \rightarrow \infty$
- Barrier problem is unbounded for fixed $\mu \implies$ iterates diverge

Handling unbounded solution sets

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} & f(x) - \mu \sum \ln(x^{(i)}) + \mu \kappa_d e^T x \\ \text{s.t.} & c(x) = 0 \end{array}$$

- What if original problem has unbounded set $\mathcal{S}$ of optimal solutions?
- Then, $\varphi_\mu(\bar{x}_l) \rightarrow -\infty$ for some $\bar{x}_l \in \mathcal{S}$ with $\bar{x}_l \rightarrow \infty$
- Barrier problem is unbounded for fixed $\mu \implies$ iterates diverge
- Remedy (from linear programming)
 - add linear damping parameter to barrier objective function
 - weight $\kappa_d \mu$ goes to zero with μ
 - corresponds to a perturbation “ $\kappa_d \mu e$ ” of the dual infeasibility in primal-dual equations
- κ_d is `kappa_d`