NTNU | Norwegian University of Science and Technology

# TTK4135 – Lecture 19
# Practical SQP algorithms for nonlinear programming

Lecturer: Lars Imsland

# Outline

- Recap: Local SQP algorithms for *equality-constrained* NLPs

- Extension to NLPs with *inequalities*

- Globalization («make it work when starting far from optimum»):
  - Computation/approximation of the Hessian of the Lagrangian
  - Linesearch

- Other issues
  - The Maratos effect
  - Infeasible linearized constraints

Reference: N&W Ch. 18.2, 18.3, 15.4

# Newton's method for solving nonlinear equations (Ch. 11)

- Solve equation system $r(x) = 0, \quad r(x) : \mathbb{R}^n \to \mathbb{R}^n$
- Assume Jacobian $J(x) \in \mathbb{R}^{n \times n}$ exists and is continuous
- Taylor: $r(x+p) = r(x) + J(x)p + O(\cancel{\|p\|^2})$

$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

**Algorithm 11.1** (Newton's Method for Nonlinear Equations).

Choose $x_0$;

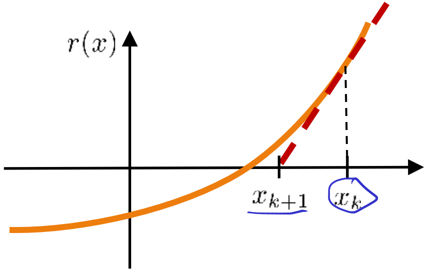
**for** $k = 0, 1, 2, \ldots$

Calculate a solution $p_k$ to the Newton equations

$$J(x_k)p_k = -r(x_k);$$

$x_{k+1} \leftarrow x_k + p_k$;

**end (for)**



# Newton's method for solving nonlinear equations (Ch. 11)

- Solve equation system $r(x) = 0, \quad r(x) : \mathbb{R}^n \to \mathbb{R}^n$
- Assume Jacobian $J(x) \in \mathbb{R}^{n \times n}$ exists and is continuous
- Taylor: $r(x + p) = r(x) + J(x)p + O(\|p\|^2)$

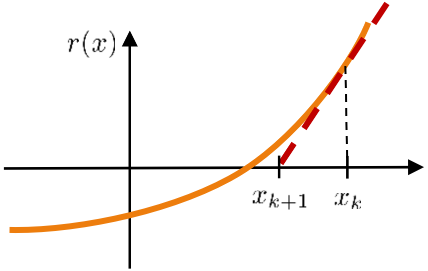
$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

**Algorithm 11.1** (Newton's Method for Nonlinear Equations).

Choose $x_0$;
**for** $k = 0, 1, 2, \ldots$
  Calculate a solution $p_k$ to the Newton equations

$$J(x_k)p_k = -r(x_k);$$

  $x_{k+1} \leftarrow x_k + p_k$;
**end (for)**




- Convergence rate (Thm 11.2): Quadratic convergence if $J(x)$ is invertible
  (quadratic convergence is very good, but only holds close to the solution)

# Newton's method for solving nonlinear equations (Ch. 11)

- Solve equation system $r(x) = 0, \quad r(x) : \mathbb{R}^n \to \mathbb{R}^n$
- Assume Jacobian $J(x) \in \mathbb{R}^{n \times n}$ exists and is continuous
- Taylor: $r(x+p) = r(x) + J(x)p + O(\|p\|^2)$

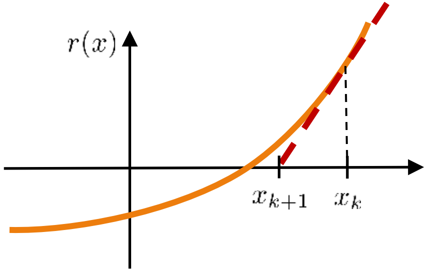
$$J(x) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

**Algorithm 11.1** (Newton's Method for Nonlinear Equations).

Choose $x_0$;
**for** $k = 0, 1, 2, \ldots$
Calculate a solution $p_k$ to the Newton equations

$$J(x_k)p_k = -r(x_k);$$

$x_{k+1} \leftarrow x_k + p_k$;
**end (for)**




- Convergence rate (Thm 11.2): Quadratic convergence if $J(x)$ is invertible (quadratic convergence is very good, but only holds close to the solution)

$\min_x f(x)$

- If we set $r(x) = \nabla f(x)$, then this method corresponds to Newton's method for minimizing $f(x)$

$\boxed{\nabla f(x) = 0}$

$$p_k = -J(x_k)^{-1} r(x_k) \longleftrightarrow p_k = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

# Equality-constrained NLPs – Newton

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \lambda^\top c(x)$

- KKT conditions: $F(x, \lambda) = \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ c(x) \end{pmatrix} = 0$

# Equality-constrained NLPs – Newton

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Lagrangian: $\mathcal{L}(x, \lambda) = f(x) - \lambda^\top c(x)$

- KKT conditions: $F(x, \lambda) = \begin{pmatrix} \nabla_x \mathcal{L}(x, \lambda) \\ c(x) \end{pmatrix} = 0$

- To solve: Use Newton's method for nonlinear equations on KKT conditions:

$$A(x)^\top = (\nabla c_1(x), \ldots, \nabla c_m(x))$$

$$\begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} + \begin{pmatrix} p_k \\ p_{\lambda_k} \end{pmatrix} \qquad \text{where} \qquad \underbrace{\begin{pmatrix} \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) & -A^\top(x_k) \\ A(x_k) & 0 \end{pmatrix}}_{\text{Jacobian of } F(x,\lambda) \text{ at } (x_k,\, \lambda_k)} \begin{pmatrix} p_k \\ p_{\lambda_k} \end{pmatrix} = \underbrace{\begin{pmatrix} -\nabla f(x_k) + A^\top(x_k)\lambda_k \\ -c(x_k) \end{pmatrix}}_{-F(x_k, \lambda_k)}$$

- Very efficient method for solving equality-constrained NLPs locally

# Equality-constrained NLP – QP

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Consider now this quadratic/linear approximation:

$$\min_{p \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^\top p + \frac{1}{2} p^\top \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) p \ \text{ subject to } \ c(x_k) + A(x_k)^\top p = 0$$

- KKT conditions:

$$\begin{pmatrix} \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) & -A^\top(x_k) \\ A(x_k) & 0 \end{pmatrix} \begin{pmatrix} p_k \\ l_k \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -c(x_k) \end{pmatrix}$$

# Equality-constrained NLP – QP

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad c(x) = 0$$

- Consider now this quadratic/linear approximation:

$$\min_{p \in \mathbb{R}^n} f(x_k) + \nabla f(x_k)^\top p + \frac{1}{2} p^\top \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) p \ \text{ subject to } \ c(x_k) + A(x_k)^\top p = 0$$

- KKT conditions:

$$\begin{pmatrix} \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) & -A^\top(x_k) \\ A(x_k) & 0 \end{pmatrix} \begin{pmatrix} p_k \\ l_k \end{pmatrix} = \begin{pmatrix} -\nabla f(x_k) \\ -c(x_k) \end{pmatrix}$$

- If we let $l_k = p_{\lambda_k} + \lambda_k = \lambda_{k+1}$, it is easy to show that the two KKT systems give equivalent solutions
  - Newton-viewpoint: quadratic convergence locally
  - QP-viewpoint: practical implementation and extension to inequality constraints

- Assumptions for the above:
  1) $A(x_k)$ full row rank (LICQ),
  2) $d^\top \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) d > 0$ for all $d \neq 0$ s.t. $A(x_k) d = 0$ ("pos.def. on tangent space of constraints")

# Local SQP-algorithm for solving NLPs

Only equality constraints:

$$\min f(x)$$
$$\text{subject to } c(x) = 0$$

**Algorithm 18.1** (Local SQP Algorithm for solving (18.1)).

Choose an initial pair $(x_0, \lambda_0)$; set $k \leftarrow 0$;
**repeat** until a convergence test is satisfied
  Evaluate $f_k$, $\nabla f_k$, $\nabla^2_{xx}\mathcal{L}_k$, $c_k$, and $A_k$;
  Solve (18.7) to obtain $p_k$ and $l_k$;
  Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow l_k$;
**end (repeat)**

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}_k p$$
$$\text{subject to} \quad A_k p + c_k = 0.$$

# Local SQP-algorithm for solving NLPs

Only equality constraints:

$$\min f(x)$$
$$\text{subject to } c(x) = 0$$

**Algorithm 18.1** (Local SQP Algorithm for solving (18.1)).

Choose an initial pair $(x_0, \lambda_0)$; set $k \leftarrow 0$;
**repeat** until a convergence test is satisfied
  Evaluate $f_k$, $\nabla f_k$, $\nabla^2_{xx}\mathcal{L}_k$, $c_k$, and $A_k$;
  Solve (18.7) to obtain $p_k$ and $l_k$;
  Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow l_k$;
**end (repeat)**

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}_k p$$
$$\text{subject to} \quad A_k p + c_k = 0.$$

Extension to inequality constraints – IQP method:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}_k p$$
$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E},$$
$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}.$$

Convergence:

- Close to the solution, a solution to the approximation has same active set as a solution to the original problem (Thm 18.1).
- Therefore, once the optimal active set is found, Algorithm 18.1 with inequalities behaves just like a problem with only equalities. That is, very good (local) convergence.
- In addition: Far from the solution, the SQP approach is usually able to improve the estimate of the active set and guide the iterates toward a solution.

# Local SQP-algorithm for solving NLPs

Only equality constraints:

$$\min f(x)$$
$$\text{subject to } c(x) = 0$$

**Algorithm 18.1** (Local SQP Algorithm for solving (18.1)).

Choose an initial pair $(x_0, \lambda_0)$; set $k \leftarrow 0$;
**repeat** until a convergence test is satisfied
  Evaluate $f_k$, $\nabla f_k$, $\nabla^2_{xx}\mathcal{L}_k$, $c_k$, and $A_k$;
  Solve (18.7) to obtain $p_k$ and $l_k$;
  Set $x_{k+1} \leftarrow x_k + p_k$ and $\lambda_{k+1} \leftarrow l_k$;
**end (repeat)**

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx}\mathcal{L}_k p$$
$$\text{subject to} \quad A_k p + c_k = 0.$$

Extension to inequality constraints – EQP method:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

Maintain/update a “working set” (approximation of the active set) in Alg. 18.1, solve equality-constrained QP in each iteration. May be more efficient for large-scale problems.

# Globalization

how to make SQP work when not starting close to the solution

# Globalization: Newton unconstrained optimization

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Quadratic approximation:

$$\min_p f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p$$

solution (Newton direction): $p_k = (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$

- Valid direction:

$\nabla^2 f(x_k) > 0 \Rightarrow p_k$ descent direction

- How to ensure valid direction:

1) Modify: use $\nabla^2 f(x_k) + \tau I$

$\rightarrow$ 2) Use Quasi-Newton approximation (BFGS)

- Line search:

$$x_{k+1} = x_k + \alpha_k p_k$$

Use $\alpha_k$ that ensures good progress in obj. function

(Wolfe cond.)

# Quasi-Newton for unconstrained problems

$$\min_{x \in \mathbb{R}^n} f(x)$$

**Algorithm 6.1** (BFGS Method).

Given starting point $x_0$, convergence tolerance $\epsilon > 0$,
inverse Hessian approximation $H_0$;

$k \leftarrow 0$;

**while** $\|\nabla f_k\| > \epsilon$;

Compute search direction

$$p_k = -H_k \nabla f_k;$$

Set $x_{k+1} = x_k + \alpha_k p_k$ where $\alpha_k$ is computed from a line search procedure to satisfy the Wolfe conditions (3.6);

Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

Compute $H_{k+1}$ by means of (6.17);

$k \leftarrow k + 1$;

**end (while)**

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

# Globalization: SQP for constrained optimization

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

- Quadratic approximation:

$p_k$ is solution to →

$$\left\{\begin{array}{l} \min_p \; f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k) p \\ \text{s.t.} \quad c_i(x_k) + \nabla c_i(x_k)^T p = 0, \quad i \in \mathcal{E} \\ \qquad\; c_i(x_k) + \nabla c_i(x_k)^T p \geq 0, \quad i \in \mathcal{I} \end{array}\right.$$

- Valid solution:

→ $\nabla^2_{xx} \mathcal{L}(x_k, \lambda_k)$ pos. def. on tangent space of constraints

- How to ensure valid solution:

1) Modify $\nabla^2_{xx} \mathcal{L}(x_k, \lambda_k)$

→ 2) Use BFGS on $\nabla^2_{xx} \mathcal{L}(x_k, \lambda_k)$

- Line search:

$x_{k+1} = x_k + \alpha_k p_k$

Use $\alpha_k$ that gives progress in <u>both</u> obj. fun. <u>and</u> constraints

How? → Merit function

# Quasi-Newton for SQP

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \tag{18.11a}$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \tag{18.11b}$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \tag{18.11c}$$

- SQP needs Hessian of Lagrangian, but this require second derivatives of objective and constraints, which may be expensive
- Quasi-Newton (BFGS) very successful for unconstrained optimization – can we do the same in the constrained case? Yes:

Unconstrained case:

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k, \tag{6.5}$$

$$\text{(BFGS)} \qquad H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \tag{6.17}$$

$$H_k \approx \left[\nabla^2 f(x_k)\right]^{-1}$$

Constrained case:

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \tag{18.13}$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}. \tag{18.16}$$

$$B_k \approx \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$$

# Quasi-Newton for SQP

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \tag{18.11a}$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \tag{18.11b}$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \tag{18.11c}$$

- SQP needs Hessian of Lagrangian, but this require second derivatives of objective and constraints, which may be expensive
- Quasi-Newton (BFGS) very successful for unconstrained optimization – can we do the same in the constrained case? Yes:

Unconstrained case:

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \qquad y_k = \nabla f_{k+1} - \nabla f_k, \tag{6.5}$$

$$\text{(BFGS)} \qquad H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \tag{6.17}$$

$$H_k \approx \left[\nabla^2 f(x_k)\right]^{-1}$$

Constrained case:

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1}). \tag{18.13}$$

$$r_k = \theta_k y_k + (1 - \theta_k) B_k s_k$$

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{r_k r_k^T}{s_k^T r_k}. \tag{18.16}$$

$$B_k \approx \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k)$$

- Possible problem: BFGS gives positive definite Hessian approximation, while Hessian of Lagrangian is not necessarily positive definite (not even close to a solution). That is, the approximation may be bad.

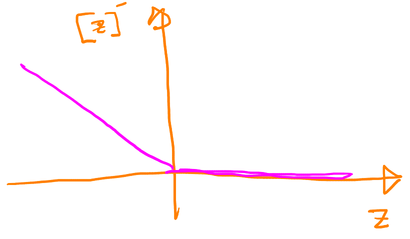
- Possible solution: Approximate "reduced Hessian" (Hessian on nullspace of constraints) instead. This reduced Hessian is much more likely to be positive definite (recall sufficient conditions).

# $l_1$ merit function – measure progress in **both** objective and constraints

$$\phi_1(x;\mu) = f(x) + \mu \sum_{i\in\mathcal{E}} |c_i(x)| + \mu \sum_{i\in\mathcal{I}} [c_i(x)]^-$$

$$[z]^- = \max\{0, -z\}$$

$\mu$ : penalty parameter




Note: $\phi_1$ is not differentiable.

## $l_1$ merit function – measure progress in both objective and constraints

Def. A merit function $\phi(x;\mu)$ is exact if,

for any $\mu > \mu^* > 0$, a local solution of NLP

is a local solution of $\phi(x;\mu)$

Thm 17.3: $\phi_1$ is exact for $\mu^* = \max\{|\lambda_i^*|, i \in \mathcal{E} \cup \mathcal{I}\}$

Implement: We don't know $\lambda_i^*$ in advance!

→ Start with $\mu = \mu_k$ small, and increase if slowly in each iteration.

# Descent direction of merit function

Assume <u>only</u> eq. constr. ($\mathcal{I} = \emptyset$): $\phi_1(x;\mu) = f(x) + \mu \|c(x)\|_1$

We need directional derivative $\underline{\nabla_x \cancel{\phi_1(x;\mu)}^T p}$

↳ not defined everywhere!

Define $D(\phi_1(x_k;\mu);p_k) := \nabla f(x_k)^T p_k - \mu \|c(x_k)\|_1$

<u>Thm 18.2</u> 1) $D(\phi_1;p_k)$ <u>is</u> directional derivative

2) $D(\phi_1;p_k) \leq -p_k^T \nabla^2_{xx} \mathcal{L}(x_k,\lambda_k) p_k - (\mu - \underline{\|\lambda_{k+1}\|_\infty}) \|c(x_k)\|_1$

That is: $p_k$ is "descent direction" if

a) $\nabla^2_{xx} \mathcal{L}(x_k,\lambda_k) > 0$ "on tangent space of constraints"

b) $\mu > \|\lambda_{k+1}\|_\infty$

# Line search on merit function

$x_{k+1} = x_k + \alpha_k p_k$

choose $\alpha_k$ s.t. $\phi_1(x_k + \alpha_k p_k ; \mu_k) \leq$

use this in backtracking line search

$\phi_1(x_k ; \mu_k) + \eta \alpha D(\phi_1(x_k ; \mu_k) ; p_k)$

constant, typically $\sim 0.1$

Compare 1st Wolfe condition:

$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha \underline{\nabla f(x_k)^T p_k}$

$\sim 10^{-4}$

# Line search – Merit function
## "Globalization"

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \qquad (18.11a)$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \qquad (18.11b)$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \qquad (18.11c)$$

- How far to walk along *p*? Linesearch (or trust region)!
- Unconstrained optimization: The Armijo (Wolfe) condition ensure sufficient decrease of objective function:

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k, \qquad (3.4)$$

- Constrained optimization: Must check both objective and constraint!
- Merit function (for line search): Function that measure progress in both:

$l_1$ merit function: $$\phi_1(x; \mu) = f(x) + \mu \sum_{i \in \mathcal{E}} |c_i(x)| + \mu \sum_{i \in \mathcal{I}} [c_i(x)]^-, \qquad (15.24)$$

$$\mu^* = \max\{|\lambda_i^*|, \; i \in \mathcal{E} \cup \mathcal{I}\}$$

**Definition 15.1** (Exact Merit Function).
*A merit function $\phi(x; \mu)$ is exact if there is a positive scalar $\mu^*$ such that for any $\mu > \mu^*$, any local solution of the nonlinear programming problem (15.1) is a local minimizer of $\phi(x; \mu)$.*

- Thm 18.2: $D(\phi_1(x_k; \mu); p_k) \leq -p_k^T \nabla_{xx}^2 \mathcal{L}_k p_k - (\mu - \|\lambda_{k+1}\|_\infty)\|c_k\|_1$
  - That is: $p_k$ is a descent direction for merit function if Hessian of Lagrangian is positive definite and $\mu$ is large enough

# A practical line search SQP method

**Algorithm 18.3** (Line Search SQP Algorithm).

Choose parameters $\eta \in (0, 0.5)$, $\tau \in (0, 1)$, and an initial pair $(x_0, \lambda_0)$;
Evaluate $f_0, \nabla f_0, c_0, A_0$;
If a quasi-Newton approximation is used, choose an initial $n \times n$ symmetric positive definite Hessian approximation $B_0$, otherwise compute $\nabla^2_{xx} \mathcal{L}_0$;
**repeat** until a convergence test is satisfied
  Compute $p_k$ by solving (18.11); let $\hat{\lambda}$ be the corresponding multiplier;
  Set $p_\lambda \leftarrow \hat{\lambda} - \lambda_k$;
  Choose $\mu_k$ to satisfy (18.36) with $\sigma = 1$;
  Set $\alpha_k \leftarrow 1$;
  **while** $\phi_1(x_k + \alpha_k p_k; \mu_k) > \phi_1(x_k; \mu_k) + \eta \alpha_k D_1(\phi(x_k; \mu_k) p_k)$
    Reset $\alpha_k \leftarrow \tau_\alpha \alpha_k$ for some $\tau_\alpha \in (0, \tau]$;
  **end (while)**
  Set $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and $\lambda_{k+1} \leftarrow \lambda_k + \alpha_k p_\lambda$;
  Evaluate $f_{k+1}, \nabla f_{k+1}, c_{k+1}, A_{k+1}$, (and possibly $\nabla^2_{xx} \mathcal{L}_{k+1}$);
  If a quasi-Newton approximation is used, set
    $s_k \leftarrow \alpha_k p_k$ and $y_k \leftarrow \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_{k+1})$,
  and obtain $B_{k+1}$ by updating $B_k$ using a quasi-Newton formula;
**end (repeat)**

$$\min_p \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p \tag{18.11a}$$
$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \tag{18.11b}$$
$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \tag{18.11c}$$

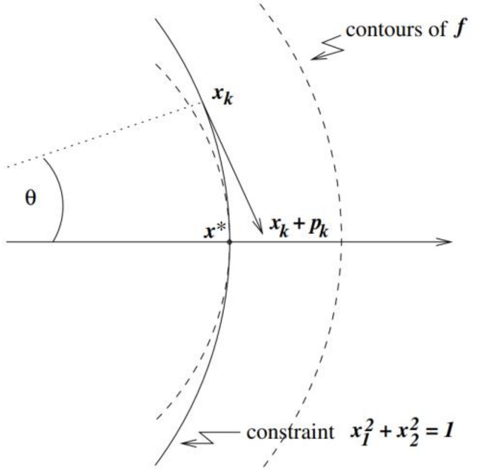
$$\mu \geq \frac{\nabla f_k^T p_k + (\sigma/2) p_k^T \nabla^2_{xx} \mathcal{L}_k p_k}{(1 - \rho) \|c_k\|_1}. \tag{18.36}$$

(choose $\mu_k$ such that $p_k$ is a descent direction for $\phi_1(x_k; \mu_k)$)

# Maratos effect

- Maratos effect: A merit function may reject good steps:

$$\min \; f(x_1, x_2) = 2(x_1^2 + x_2^2 - 1) - x_1, \quad \text{subject to} \quad x_1^2 + x_2^2 - 1 = 0. \qquad (15.34)$$

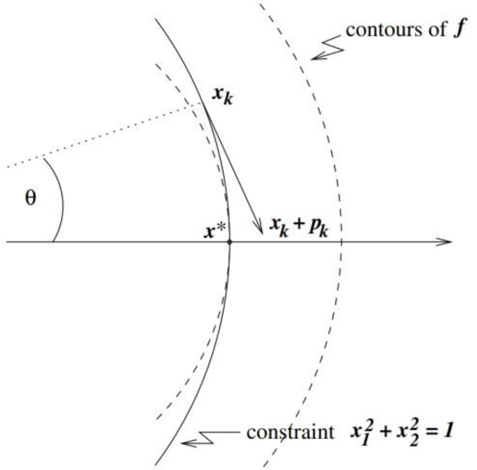



$p_k$ good step even if both objective and constraint violation increase!

# Maratos effect

- Maratos effect: A merit function may reject good steps:

$$\min\ f(x_1, x_2) = 2(x_1^2 + x_2^2 - 1) - x_1, \quad \text{subject to} \quad x_1^2 + x_2^2 - 1 = 0. \qquad (15.34)$$




$p_k$ good step even if both objective and constraint violation increase!

- Remedy:
  - Use a merit function that does not suffer from the Maratos effect
  - Use “non-monotone” strategy (temporarily allow increase in merit function)
  - Use “second-order correction” (when Maratos effect occurs)

# Inconsistent linearizations

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases}$$

In each SQP iteration, solve:

$$\min_{p} \quad f_k + \nabla f_k^T p + \tfrac{1}{2} p^T \nabla_{xx}^2 \mathcal{L}_k p \tag{18.11a}$$

$$\text{subject to} \quad \nabla c_i(x_k)^T p + c_i(x_k) = 0, \quad i \in \mathcal{E}, \tag{18.11b}$$

$$\nabla c_i(x_k)^T p + c_i(x_k) \geq 0, \quad i \in \mathcal{I}. \tag{18.11c}$$

Ex.

$$\left.\begin{array}{lll} c_1(x) = -x + 1 \geq 0 & \rightarrow & x < 1 \\ c_2(x) = x^2 - 4 \geq 0 & \rightarrow & x \geq 2 \text{ or } x \leq -2 \end{array}\right\} \text{consistent}$$

Linearize at $x_k = 1$:

$$\left.\begin{array}{lll} c_1(x_k) + \nabla c_1(x_k)^T p = -p \geq 0 & \rightarrow & p \leq 0 \\ c_2(x_k) + \nabla c_2(x_k)^T p = -3 + 2p \geq 0 & \rightarrow & p \geq \frac{2}{3} \end{array}\right\} \text{Inconsistent}$$

Practical solution: Relax constraints

$$\min_{x, t} \quad f(x) + \mu(t_1 + t_2)$$

$$\text{s.t.} \quad c_1(x) \geq 0 - t_1$$

$$c_2(x) \geq 0 - t_2, \quad t_1, t_2 \geq 0$$

# NLP software

- SNOPT
  - "solves large-scale linear and nonlinear problems; especially recommended if some of the constraints are highly nonlinear, or constraints respectively their gradients are costly to evaluate and second derivative information is unavailable or hard to obtain; assumes that the number of "free" variables is modest."
  - Licence: Commercial
- IPOPT
  - "interior point method for large-scale NLPs"
  - License: Open source
- WORHP
  - SQP solver for very large problems, IP at QP level, exact or approximate second derivatives, various linear algebra options, varius interfaces
  - Licence: Commercial, but free for academia
- KNITRO
  - trust region interior point method, efficient for NLPs of all sizes, various interfaces
  - License: Commercial

- (…and several others, including `fmincon` in Matlab Optimization Toolbox)

- «Decision tree for optimization software»: http://plato.asu.edu/sub/nlores.html

# Example: optimization using CasADi

- CasADi (https://casadi.org/)
  - "CasADi is a symbolic framework for numeric optimization implementing automatic differentiation in forward and reverse modes on sparse matrix-valued computational graphs."

$$\min_{x,y,z} \; x^2 + 100z^2$$
$$\text{s.t.} \;\; z + (1-x)^2 - y = 0$$

Define variables

Define objective and constraints

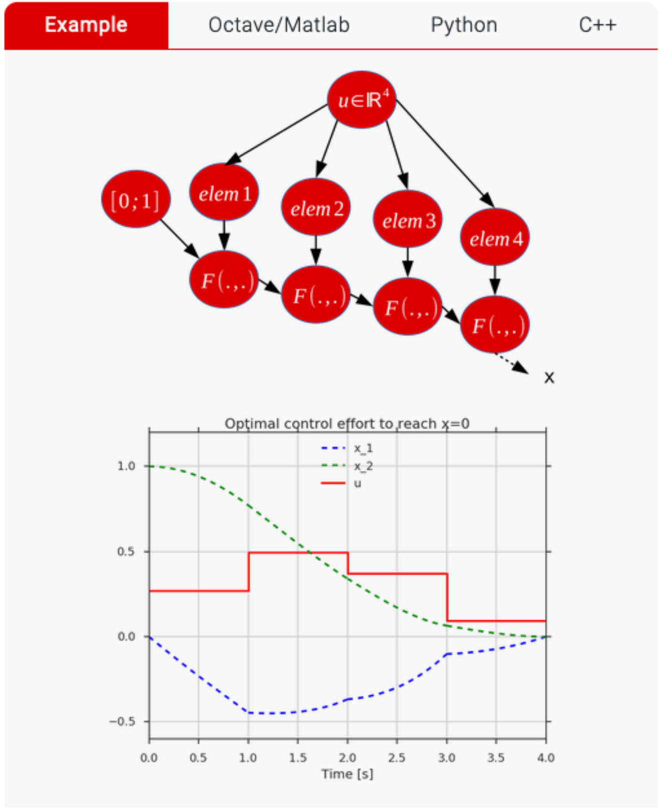
Create solver object

Solve the opt problem

**rosenbrock.m**

```
import casadi.*

% Create NLP: Solve the Rosenbrock problem:
%     minimize    x^2 + 100*z^2
%     subject to  z + (1-x)^2 - y == 0
x = SX.sym('x');
y = SX.sym('y');
z = SX.sym('z');
v = [x;y;z];
f = x^2 + 100*z^2;
g = z + (1-x)^2 - y;
nlp = struct('x', v, 'f', f, 'g', g);

% Create IPOPT solver object
solver = nlpsol('solver', 'ipopt', nlp);

% Solve the NLP
res = solver('x0' , [2.5 3.0 0.75],... % solution guess
             'lbx', -inf,...           % lower bound on x
             'ubx',  inf,...           % upper bound on x
             'lbg',    0,...           % lower bound on g
             'ubg',    0);             % upper bound on g

% Print the solution
f_opt = full(res.f)           % >> 0
x_opt = full(res.x)           % >> [0; 1; 0]
lam_x_opt = full(res.lam_x)   % >> [0; 0; 0]
lam_g_opt = full(res.lam_g)   % >> 0
```

# Example from CasADi




Example | Octave/Matlab | Python | C++

```python
from casadi import *


x = MX.sym('x',2); # Two states
p = MX.sym('p');   # Free parameter

# Expression for ODE right-hand side
z = 1-x[1]**2;
rhs = vertcat(z*x[0]-x[1]+2*tanh(p),x[0])

# ODE declaration with free parameter
ode = {'x':x,'p':p,'ode':rhs}

# Construct a Function that integrates over 1s
F = integrator('F','cvodes',ode,{'tf':1})

# Control vector
u = MX.sym('u',4,1)

x = [0,1]  # Initial state
for k in range(4):
  # Integrate 1s forward in time:
  # call integrator symbolically
  res = F(x0=x,p=u[k])
  x = res["xf"]


# NLP declaration
nlp = {'x':u,'f':dot(u,u),'g':x};

# Solve using IPOPT
solver = nlpsol('solver','ipopt',nlp)
res = solver(x0=0.2,lbg=0,ubg=0)

plot(res["x"])
```