Fast Model Predictive Control for Magnetic Plasma Control
MPC using fast online 1st-order QP methods

Matija Perne
Jožef Stefan Institute
Linear dynamic system, optimum

Discrete-time model:

\[ x_{k+1} = Ax_k + Bu_k \]
\[ y_k = Cx_k + Du_k \]
• Discrete-time model:

\[ x_{k+1} = A x_k + B u_k \]

\[ y_k = C x_k \]
Discrete-time model:

\[ x_{k+1} = Ax_k + Bu_k \]

Cost minimization:

\[ J_t = \frac{1}{2} (x - x_r)' Q_t (x - x_r) + \frac{1}{2} (u - u_r)' R_t (u - u_r) \]
Linear dynamic system, optimum

\[ x_{k+1} = A x_k + B u_k \]

- Discrete-time model:
  \[ y_k = C x_k \]

- Cost minimization:
  \[ J_t = \frac{1}{2} (x - x_r)' Q_t (x - x_r) + \frac{1}{2} (u - u_r)' R_t (u - u_r) \]

  \[ u \in \mathcal{U} \quad u_{min} \leq u \leq u_{max} \]

- Constraints:
  \[ x \in \mathcal{X} \quad x_{min} \leq x \leq x_{max} \]

  \[ y \in \mathcal{Y} \quad y_{min} \leq y \leq y_{max} \]
Linear dynamic system, optimum

\[ u \rightarrow \mathbf{x} \rightarrow y \]

- Discrete-time model:
  \[ x_{k+1} = A x_k + B u_k \]
  \[ y_k = C x_k \]

- Cost minimization:
  \[ J_t = \frac{1}{2} (x - x_r)' Q_t (x - x_r) + \frac{1}{2} (u - u_r)' R_t (u - u_r) \]

- Constraints:
  \[ u_{\text{min}} \leq U u \leq u_{\text{max}} \]
  \[ x_{\text{min}} \leq X x \leq x_{\text{max}} \]
  \[ y_{\text{min}} \leq Y y \leq y_{\text{max}} \]
Planning the future

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]
Planning the future

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]

\[ u^* = \arg \min_u J \]
Planning the future

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]

\[ x_{k+1} = A x_k + B u_k \]

\[ u^* = \arg \min_u J \quad \text{s.t.} \quad x_{k+1} \in \mathcal{X} \]

\[ u_k \in \mathcal{U} \]
Quadratic program

\[ J = z' H z + g' z \]

\[ z = \begin{bmatrix}
    x_0 \\
    \vdots \\
    x_{N-1} \\
    u_0 \\
    \vdots \\
    u_{N-1}
\end{bmatrix} \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]
Quadratic program

\[ J = z' H z + g' z \]

\[ z = \begin{bmatrix}
  x_0 \\
  \vdots \\
  x_{N-1} \\
  u_0 \\
  \vdots \\
  u_{N-1}
\end{bmatrix} \]

Hessian

\[ A_C z = b_C \]

\[ z \in \mathbb{Z} \]
Quadratic program

\[ z = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \\ u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \]

\[ J = z' H z + g' z \]

Hessian

Gradient

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]
Quadratic program

\[ z = \begin{bmatrix}
  x_0 \\
  \vdots \\
  x_{N-1} \\
  u_0 \\
  \vdots \\
  u_{N-1}
\end{bmatrix} \]

\[ J = z' H z + g' z \]

Hessian

Affine set

Gradient

\[ A_c z = b_c \]

\[ z \in \mathcal{Z} \]
Quadratic program

\[
J = z' H z + g' z
\]

Hessian
Gradient

Affine set

\[
A_c z = b_c
\]

Polyhedron

\[
z \in \mathbb{Z}
\]
Quadratic program

\[ z = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \\ u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \]

\[ J = z' \mathbf{H} z + g' z \]

Hessian

Gradient

Affine set

Polyhedron

Dual method

\[ A_C z = b_C \]

\[ z \in \mathcal{Z} \]
• Explicit MPC: solving the problem in advance
• Explicit MPC: solving the problem in advance
  – Solution piecewise linear
Explicit MPC: solving the problem in advance

Solution piecewise linear

(http://www.seas.upenn.edu/~ese680/papers/explicit_linear_mpc.pdf)
• Explicit MPC: solving the problem in advance
  – Solution piecewise linear
  – Active and inactive constraints:
    \[ a \leq b \]
    (http://www.seas.upenn.edu/~ese680/papers/explicit_linear_mpc.pdf)
Explicit MPC: solving the problem in advance
- Solution piecewise linear
- Active and inactive constraints:

\[ a \leq b \]
\[ a < b \]

(http://www.seas.upenn.edu/~ese680/papers/explicit_linear_mpc.pdf)
• Explicit MPC: solving the problem in advance
  - Solution piecewise linear
  - Active and inactive constraints:

\[
a \leq b \\
\quad a < b \\
\quad a = b
\]

(http://www.seas.upenn.edu/~ese680/papers/explicit_linear_mpc.pdf)
• Explicit MPC: solving the problem in advance
  – Solution piecewise linear
  – Active and inactive constraints:

\[
a \leq b
\]

\[
a < b \quad a = b
\]

• Active set methods
Solving the QP

- Explicit MPC: solving the problem in advance
  - Solution piecewise linear
  - Active and inactive constraints:
    
    \[
    a \leq b
    \]

- Active set methods
  - Online search for active constraints
Solving the QP

- Explicit MPC: solving the problem in advance
  - Solution piecewise linear
  - Active and inactive constraints:
    \[ a \leq b \]
    \[ a < b \]
    \[ a = b \]

- Active set methods
  - Online search for active constraints
  - Iterative
• Interior point methods
• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
  – Modified problem solved with unconstrained optimization method
• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
  – Modified problem solved with unconstrained optimization method
  – Modification decreased iteratively
• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
  – Modified problem solved with unconstrained optimization method
  – Modification decreased iteratively

• First order methods
Solving the QP

• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
  – Modified problem solved with unconstrained optimization method
  – Modification decreased iteratively

• First order methods
  – Inspired by gradient method
Solving the QP

• Interior point methods
  – Cost is modified with a steep smooth function close to polyhedral constraints
  – Modified problem solved with unconstrained optimization method
  – Modification decreased iteratively

• First order methods
  – Inspired by gradient method
  – Simple, low order of convergence
Existence of a solution

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]

\[ u^* = \arg \min_u J \quad \text{s.t.} \quad x_{k+1} = A x_k + B u_k \]

\[ x_{k+1} \in \mathcal{X} \]

\[ u_k \in \mathcal{U} \]
Existence of a solution

\[
J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k
\]

\[
x_{k+1} = A x_k + B u_k
\]

\[
x_{k+1} \in \mathcal{X}
\]

\[
u_k \in \mathcal{U}
\]

\[u^* = \arg \min_u J \quad \text{s.t.} \quad x_{k+1} \in \mathcal{X}, \quad u_k \in \mathcal{U}\]
Existence of a solution

\[
J = z' H z + g' z
\]

\[
A_c z = b_c
\]

\[
z \in \mathbb{Z}
\]
Existence of a solution

\[ z = \begin{bmatrix} x_0 \\ \vdots \\ x_{N-1} \\ u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \]

\[ J = z'Hz + g'z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]

Soft constraints
Making the QP convenient

\[
\begin{bmatrix}
  x_0 \\
  \vdots \\
  x_{N-1} \\
  u_0 \\
  \vdots \\
  u_{N-1}
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x_0 \\
  u_0 \\
  x_1 \\
  u_1 \\
  \vdots \\
  x_{N-1} \\
  u_{N-1}
\end{bmatrix}
\]

\[
z = \begin{bmatrix}
  u_0 \\
  u_1 \\
  \vdots \\
  u_{N-1}
\end{bmatrix}
\]

\[
z' = P \cdot z
\]
Simplifying the QP

Move blocking, constraint reduction

\[ J = z' H z + g' z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]
Move blocking, constraint reduction

\[ J = z' H z + g' z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]
Simplifying the QP

Move blocking, constraint reduction

\[ J = z'Hz + g'z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]
Simplifying the QP

Move blocking, constraint reduction

\[ J = z' H z + g' z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]

\[ z = \begin{bmatrix}
  x_0 \\
  \vdots \\
  x_{N-1} \\
  u_0 \\
  \vdots \\
  u_{N-1}
\end{bmatrix} \]

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x_k' Q x_k + g_x' x_k + \frac{1}{2} u_k' R u_k + g'_u u_k \]
Move blocking, constraint reduction

\[ J = z' H z + g' z \]

\[ A_c z = b_c \]

\[ z \in \mathbb{Z} \]

\[ J = \sum_{k=0}^{N-1} \frac{1}{2} x'_k Q x_k + g'_x x_k + \frac{1}{2} u'_k R u_k + g'_u u_k \]
Simplifying the QP

Move blocking, constraint reduction

\[
J = z' \mathbf{H} z + g' z
\]

\[
A_c z = b_c
\]

\[\begin{bmatrix}
x_0 \\
\vdots \\
x_{N-1} \\
u_0 \\
\vdots \\
u_{N-1}
\end{bmatrix}
\]

\[
x_{k+1} = \mathbf{A} x_k + \mathbf{B} u_k
\]

\[x_{k+1} \in \mathcal{X}\]

\[u_k \in \mathcal{U}\]

\[
J = \sum_{k=0}^{N-1} \frac{1}{2} x_k' \mathbf{Q} x_k + g_x' x_k + \frac{1}{2} u_k' \mathbf{R} u_k + g'_u u_k
\]
Cat

(http://animalswalls.blogspot.si/2011/08/cat-wallpapers.html)
Gradient method (ordinary)

Algorithm 4.1 Gradient method for smooth convex optimization

**Input:** Initial iterate \( z^0 \in \mathbb{R}^s \), Lipschitz constant \( L \) of \( \nabla f \)

**Output:** Approximate minimizer

\[
\text{repeat} \quad z^{k+1} = z^k - \frac{1}{L} \nabla f(z^k) \\
\text{until} \text{ stopping criterion is satisfied}
\]

unconstrained

Algorithm 4.6 Gradient method for constrained smooth convex optimization

**Input:** Initial iterate \( z^0 \in S \), Lipschitz constant \( L \) of \( \nabla f \)

**Output:** Approximate minimizer

\[
\text{repeat} \quad z^{k+1} = \pi_S \left( z^k - \frac{1}{L} \nabla f(z^k) \right) \\
\text{until} \text{ stopping criterion is satisfied}
\]

constrained
**Lipschitz constant**

**Figure 4.1** Upper and lower bounds on functions: The quadratic upper bound stems from $L$-smoothness of $f$ (Definition 4.2), while the quadratic lower bound is obtained from strong convexity of $f$ (Theorem 4.1). In this example, $f(z) = z^2 + \frac{1}{2}z + 1$ and we have chosen $L = 3$ and $\mu = 1$ for illustration.
Gradient method (ordinary)

(F. Borrelli, A. Bemporad, M. Morari, Predictive Control for linear and hybrid systems, 2015.)

unconstrained

\textbf{Algorithm 4.1} Gradient method for smooth convex optimization

\textbf{Input}: Initial iterate $z^0 \in \mathbb{R}^n$, Lipschitz constant $L$ of $\nabla f$

\textbf{Output}: Approximate minimizer

\begin{verbatim}
repeat
  $z^{k+1} = z^k - \frac{1}{L} \nabla f(z^k)$
until stopping criterion is satisfied
\end{verbatim}

constrained

\textbf{Algorithm 4.6} Gradient method for constrained smooth convex optimization

\textbf{Input}: Initial iterate $z^0 \in S$, Lipschitz constant $L$ of $\nabla f$

\textbf{Output}: Approximate minimizer

\begin{verbatim}
repeat
  $z^{k+1} = \pi_S \left( z^k - \frac{1}{L} \nabla f(z^k) \right)$
until stopping criterion is satisfied
\end{verbatim}
Gradient method (fast)

(F. Borrelli, A. Bemporad, M. Morari, Predictive Control for linear and hybrid systems, 2015.)

**Algorithm 4.2** Fast gradient method for smooth convex optimization

**Input:** Initial iterates \( z^0 \in \mathbb{R}^s, y^0 = z^0; \alpha^0 = \frac{1}{2}(\sqrt{5} - 1) \), Lipschitz constant \( L \) of \( \nabla f \)

**Output:** Approximate minimizer

```
repeat
  \[ z^{k+1} = y^k - \frac{1}{L} \nabla f(y^k) \]
  \[ \alpha^{k+1} = \frac{\alpha^k}{2} \left( \sqrt{\alpha^k^2 + 4} - \alpha^k \right) \]
  \[ \beta^k = \frac{\alpha^k(1-\alpha^k)}{\alpha^k^2 + \alpha^k + 1} \]
  \[ y^{k+1} = z^{k+1} + \beta^k(z^{k+1} - z^k) \]
until stopping criterion is satisfied
```

---

unconstrained

---

**Algorithm 4.7** Fast gradient method for constrained smooth strongly convex optimization

**Input:** Initial iterates \( z^0 \in S, y^0 = z^0; 0 < \sqrt{\mu/L} \leq \alpha^0 < 1 \), Lipschitz constant \( L \) of \( \nabla f \), strong convexity parameter \( \mu \) of \( f \)

**Output:** Approximate minimizer

```
repeat
  \[ z^{k+1} = \pi_S\left( y^k - \frac{1}{L} \nabla f(y^k) \right) \]
  \[ \alpha^{k+1} \in (0, 1): \alpha^{k+1} = (1 - \alpha^{k+1})\alpha^k + \frac{\mu \alpha^{k+1}}{L} \]
  \[ \beta^k = \frac{\alpha^k(1-\alpha^k)}{\alpha^k^2 + \alpha^k + 1} \]
  \[ y^{k+1} = z^{k+1} + \beta^k(z^{k+1} - z^k) \]
until stopping criterion is satisfied
```

---

constrained
\[
\max_x f(x) \quad \text{s.t.} \quad g(x) = 0
\]
Lagrange multipliers

\[
\max_x f(x) \quad \text{s.t.} \quad g(x) = 0
\]

\[
\mathcal{L}(x, \lambda) := f(x) + \lambda \cdot g(x)
\]
Lagrange multipliers

\[
\max_x f(x) \quad \text{s.t.} \quad g(x) = 0
\]

\[
\mathcal{L}(x, \lambda) := f(x) + \lambda \cdot g(x)
\]

\[
\nabla_{x, \lambda} \mathcal{L}(x, \lambda) = 0
\]
minimize \quad f_0(x)
subject to \quad f_i(x) \leq 0, \quad i = 1, \ldots, m
\quad h_i(x) = 0, \quad i = 1, \ldots, p,
minimize \quad f_0(x) \\
subject to \quad f_i(x) \leq 0, \quad i = 1, \ldots, m \\
\quad h_i(x) = 0, \quad i = 1, \ldots, p,

\[ L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x), \]
Lagrange and dual functions

\[ g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left( f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \]
Lagrange and dual functions

\[ g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left( f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \]

\[ g(\lambda, \nu) \leq P^* \quad \lambda \geq 0 \]
\[ g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu) = \inf_{x \in \mathcal{D}} \left( f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \]

\[ g(\lambda, \nu) \leq p^* \quad \lambda \geq 0 \quad \text{maximize} \quad g(\lambda, \nu) \]

subject to \quad \lambda \geq 0
Lagrange and dual functions

\[
g(\lambda, \nu) = \inf_{x \in D} L(x, \lambda, \nu) = \inf_{x \in D} \left( f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right)
\]

\[
g(\lambda, \nu) \leq p^* \quad \lambda \geq 0 \quad \text{maximize} \quad g(\lambda, \nu) \quad \text{subject to} \quad \lambda \geq 0
\]

We mentioned at the beginning of §5.5.3 that if strong duality holds and a dual optimal solution \((\lambda^*, \nu^*)\) exists, then any primal optimal point is also a minimizer of \(L(x, \lambda^*, \nu^*)\). This fact sometimes allows us to compute a primal optimal solution from a dual optimal solution.
(generalized fast dual gradient method)

Improving Fast Dual Ascent for MPC - Part II: The Embedded Case*

Pontus Giselsson*

* Electrical Engineering, Stanford University
  (e-mail: pontusg@stanford.edu).
(generalized fast dual gradient method)

Improving Fast Dual Ascent for MPC - Part II: The Embedded Case

Pontus Giselsson*

* Electrical Engineering, Stanford University
(e-mail: pontusg@stanford.edu).

minimize \( \frac{1}{2} y^T H y \)
subject to
\( Ay = b \bar{x} \)
\( By = v \)
\( d \leq v \leq \bar{d} \)

Letting \( H_A = AH^{-1}A^T \), the algorithm becomes
\[
y^k = H^{-1}(A^T H_A^{-1}(AH^{-1}B^T v^k + b\bar{x}) - B^T v^k) \quad (47)
\]
\[
\mu^k = \text{prox}_{g^*}^L(v^k + L^{-1}_\mu B y^k) \quad (48)
\]
\[
t^{k+1} = \frac{1 + \sqrt{1 + 4(t^k)^2}}{2} \quad (49)
\]
\[
v^{k+1} = \mu^k + \left( \frac{t^k - 1}{t^{k+1}} \right) (\mu^k - \mu^{k-1}) \quad (50)
\]
(generalized fast dual gradient method)

Improving Fast Dual Ascent for MPC -
Part II: The Embedded Case *

Pontus Giselsson *

* Electrical Engineering, Stanford University
(e-mail: pontusg@stanford.edu).

minimize \( \frac{1}{2} y^T H y \)
subject to
\[ Ay = b \bar{x} \]
\[ By = v \]
\[ d \leq v \leq \bar{d} \]

Letting \( H_A = AH^{-1}A^T \), the algorithm becomes
\[
y^k = H^{-1}(A^T H_A^{-1}(AH^{-1}B^Tv^k + b\bar{x}) - B^Tv^k) \quad (47)
\]
\[
\mu^k = \operatorname{prox}_{g^*}^{L} (v^k + L^{-1}By^k) \quad (48)
\]
\[
t^{k+1} = \frac{1 + \sqrt{1 + 4(t^k)^2}}{2} \quad (49)
\]
\[
v^{k+1} = \mu^k + \left( \frac{t^{k-1}}{t^{k+1}} \right)(\mu^k - \mu^{k-1}) \quad (50)
\]

\[ \operatorname{prox}_{\psi}^{L}(x) := \arg \min_{y} \{ \psi(y) + \frac{1}{2}\|y - x\|_L^2 \} \]

\[ \operatorname{prox}_{g^*}^{L}(x) + L^{-1}\operatorname{prox}_{g}^{L^{-1}}(Lx) = x \]
• Code generator:
• Code generator:
  - Problem described in MATLAB
• Code generator:
  - Problem described in MATLAB
  - MATLAB does the offline calculations
● Code generator:
  - Problem described in MATLAB
  - MATLAB does the offline calculations
  - MATLAB outputs the algorithm code in C
• **Code generator:**
  - Problem described in MATLAB
  - MATLAB does the offline calculations
  - MATLAB outputs the algorithm code in C
  - Code compiled, ran, result returned to MATLAB
• Code generator:
  − Problem described in MATLAB
  − MATLAB does the offline calculations
  − MATLAB outputs the algorithm code in C
  − Code compiled, ran, result returned to MATLAB
  − FGMdual one of the supported algorithms
**QPgen**

**Code generator:**
- Problem described in MATLAB
- MATLAB does the offline calculations
- MATLAB outputs the algorithm code in C
- Code compiled, ran, result returned to MATLAB
- FGMdual one of the supported algorithms

**Knowing the code:**
QPgen

• Code generator:
  − Problem described in MATLAB
  − MATLAB does the offline calculations
  − MATLAB outputs the algorithm code in C
  − Code compiled, ran, result returned to MATLAB
  − FGMdual one of the supported algorithms

• Knowing the code:
  − MATLAB code is long
• Code generator:
  - Problem described in MATLAB
  - MATLAB does the offline calculations
  - MATLAB outputs the algorithm code in C
  - Code compiled, ran, result returned to MATLAB
  - FGMdual one of the supported algorithms

• Knowing the code:
  - MATLAB code is long
  - Generated code is confusing
```c
while ((jj < *max_iter) && (cond < 0)) {
    jj++;
    copy_vec_part_negate(v, tmp_var_p, 40);
    mat_vec_mult_sparse(&CT, tmp_var_p, tmp_var_n);
    vec_sub(tmp_var_n, q1, tmp_var_n, 60);
    stack_vec(tmp_var_n, q2, rhs, 60, 40);
    perm_fwdsolve(&L, p, rhs, tmp_var_nm);
    mat_vec_mult_sparse(&Dinv, tmp_var_nm, tmp_var_nm2);
    backsolve_perm(&LT, p, tmp_var_nm2, tmp_var_nm);
    copy_vec_part(tmp_var_nm, x, 60);
    mat_vec_mult_sparse(&C, x, tmp_var_p);
    vec_add(v, tmp_var_p, tmp_var_p, 40);
    copy_vec_part(tmp_var_p, arg_prox_h, 40);
    clip_soft(tmp_var_p, l, u, (double *) &soft, 40);
    mat_vec_mult_diag(&Dinv, tmp_var_p, y);
    copy_vec_part(lambda, lambda_old, 40);
    vec_sub(arg_prox_h, tmp_var_p, lambda, 40);
    vec_sub(lambda, lambda_old, tmp_var_p, 40);
    theta_old = theta;
    theta = (1 + sqrt(1 + 4 * pow(theta_old, 2))) / 2;
    scalar_mult((theta_old - 1) / theta, tmp_var_p, 40);
    copy_vec_part(v, v_old, 40);
    vec_add(tmp_var_p, v_old, tmp_var_p, 40);

    if (mod(jj, 10) == 0) {
        cond = check_stop_cond_FGM(&Dinv, lambda, lambda_old, tmp_var_p, tm
    }
    restart(lambda, lambda_old, v, v_old, tmp_var_p, tmp_var_p2, 40);
}
```

- \( tmp\_var\_p = -v^k \)
- \( tmp\_var\_n = B^T tmp\_var\_p = -B^T v^k \)
- \( tmp\_var\_n = tmp\_var\_n - g = -B^T v^k - g \)
- \( rhs = \begin{bmatrix} tmp\_var\_n \\ b \end{bmatrix} = \begin{bmatrix} -B^T v^k - g \\ b \end{bmatrix} \)
- \( L tmp\_var\_nm = rhs, \ tmp\_var\_nm = L^{-1} rhs \) \( (K = LDL^T) \)
- \( tmp\_var\_nm2 = D^{-1} tmp\_var\_nm \) \( (D^{-1} L^{-1} rhs) \)
- \( L^T tmp\_var\_nm = tmp\_var\_nm2, \ tmp\_var\_nm = (L^T)^{-1} D^{-1} L^{-1} rhs \)
- \( \begin{bmatrix} y \\ \xi \end{bmatrix} = tmp\_var\_nm \)
- \( tmp\_var\_p = By \)
- \( tmp\_var\_p = tmp\_var\_p + v^k = By + v^k \)
- \( arg\_prox\_h = tmp\_var\_p = By + v^k \)
- \( tmp\_var\_p = \max(\min(tmp\_var\_p, u), l) \)
- \( \mu^{k-1} = \mu^k \)
- \( \mu^k = arg\_prox\_h - tmp\_var\_p \)
- \( tmp\_var\_p = \mu^k - \mu^{k-1} \)
- \( t^k = t^{k+1} \)
- \( t^{k+1} = \frac{1 + \sqrt{1 + 4 (t^k)^2}}{2} \)
- \( tmp\_var\_p = \left( \frac{t^k - 1}{t^{k+1}} \right) tmp\_var\_p = \left( \frac{t^k - 1}{t^{k+1}} \right) (\mu^k - \mu^{k-1}) \)
- \( v^{k-1} = v^k \)
- \( v^k = tmp\_var\_p + \mu^k \left( = \mu^k + \left( \frac{t^k - 1}{t^{k+1}} \right) (\mu^k - \mu^{k-1}) \right) \)

dokler pogoji niso izpolnjeni
Letting $H_A = A^T H^{-1} A$, the algorithm begins with
\[ y^k = H^{-1} (A^T H^{-1} (A H^{-1} B^T v^k + b) - B^T v^k) \]
\[ \mu^k = \text{prox}_g (v^k + L_{\mu}^{-1} B y^k) \]
\[ t^{k+1} = \frac{1 + \sqrt{1 + 4 (t^k)^2}}{2} \]
\[ v^{k+1} = \mu^k + \left( t^{k-1} t^{k+1} \right) (\mu^k - \mu^{k-1}) \]

\[
\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} y^k \\ \xi \end{bmatrix} = \begin{bmatrix} -B^T v^k \\ b \bar{x} \end{bmatrix}
\]

prox$_g^L (x) + L^{-1} \text{prox}_g (Lx) = x$

prox$_g^L (x) := \arg \min_y \left\{ \psi(y) + \frac{1}{2} \| y - x \|_L^2 \right\}$
\[\begin{align*}
\text{tmp.var.p} &= -v^k \\
\text{tmp.var.n} &= B^T \text{tmp.var.p} \ (= -B^T v^k) \\
\text{tmp.var.n} &= \text{tmp.var.n} - g \ (= -B^T v^k - g) \\
\text{rhs} &= \begin{bmatrix} \text{tmp.var.n} \\ b\bar{x} \end{bmatrix} \ (= \begin{bmatrix} -B^T v^k - g \\ b\bar{x} \end{bmatrix}) \\
L\text{tmp.var.nm} &= \text{rhs}, \quad \text{tmp.var.nm} = L^{-1} \text{rhs} \quad (K = LDL^T) \\
\text{tmp.var.nm2} &= D^{-1} \text{tmp.var.nm} \ (= D^{-1} L^{-1} \text{rhs}) \\
L^T \text{tmp.var.nm} &= \text{tmp.var.nm2}, \quad \text{tmp.var.nm} = (L^T)^{-1} D^{-1} L^{-1} \text{rhs} \\
\end{align*}\]

\[
\begin{bmatrix} y \\ \xi \end{bmatrix} = \text{tmp.var.nm}
\]

tmp.var.p = By

tmp.var.p = tmp.var.p + v^k \ (= By + v^k)

arg.prox.h = tmp.var.p \ (= By + v^k)

\[
\text{tmp.var.p} = \max (\min (\text{tmp.var.p}, u), l)
\]

\[
\mu^{k-1} = \mu^k
\]

\[
\mu^k = \text{arg.prox.h} - \text{tmp.var.p}
\]

\[
\text{tmp.var.p} = \mu^k - \mu^{k-1}
\]

\[
t^k = t^{k+1}
\]

\[
t^{k+1} = \frac{1 + \sqrt{1 + 4(t^k)^2}}{2}
\]

\[
\text{tmp.var.p} = \left(\frac{t^k - 1}{t^{k+1}}\right) \text{tmp.var.p} \left(= \left(\frac{t^k - 1}{t^{k+1}}\right) (\mu^k - \mu^{k-1})\right)
\]

\[
v^{k-1} = v^k
\]

\[
v^k = \text{tmp.var.p} + \mu^k \left(= \mu^k + \left(\frac{t^k - 1}{t^{k+1}}\right) (\mu^k - \mu^{k-1})\right)
\]

dokler pogoji niso izpolnjeni
\[ \text{tmp.var.p} = -v^k \]
\[ \text{tmp.var.n} = B^T \text{tmp.var.p} (= \text{B}^T v^k) \]
\[ \text{tmp.var.n} = \text{tmp.var.n} - g (= \text{B}^T v^k - g) \]
\[ \text{rhs} = \begin{bmatrix} \text{tmp.var.n} \\ b\bar{x} \end{bmatrix} = \begin{bmatrix} -\text{B}^T v^k - g \\ b\bar{x} \end{bmatrix} \]
\[ L \text{tmp.var.nm} = \text{rhs}, \quad \text{tmp.var.nm} = L^{-1} \text{rhs} \quad (K = L D L^T) \]
\[ \text{tmp.var.nm2} = D^{-1} \text{tmp.var.nm} (= D^{-1} L^{-1} \text{rhs}) \]
\[ L^T \text{tmp.var.nm} = \text{tmp.var.nm2}, \quad \text{tmp.var.nm} = (L^T)^{-1} D^{-1} L^{-1} \text{rhs} \]
\[ \begin{bmatrix} y \\ \xi \end{bmatrix} = \text{tmp.var.nm} \]
\[ \text{tmp.var.p} = By \]
\[ \text{tmp.var.p} = \text{tmp.var.p} + v^k (= By + v^k) \]
\[ \text{arg.prox.h} = \text{tmp.var.p} (= By + v^k) \]
\[ \textbf{tmp.var.p} = \max (\min (\text{tmp.var.p}, u), l) \]
\[ \mu^{k-1} = \mu^k \]
\[ \mu^k = \text{arg.prox.h} - \text{tmp.var.p} \]
\[ \text{tmp.var.p} = \mu^k - \mu^{k-1} \]
\[ t^k = t^{k+1} \]
\[ t^{k+1} = \frac{1 + \sqrt{1 + 4(t^k)^2}}{2} \]
\[ \text{tmp.var.p} = \left( \frac{t^k - 1}{t^{k+1}} \right) \text{tmp.var.p} = \left( \frac{t^k - 1}{t^{k+1}} \right) (\mu^k - \mu^{k-1}) \]
\[ v^{k-1} = v^k \]
\[ v^k = \text{tmp.var.p} + \mu^k (= \mu^k + \left( \frac{t^k - 1}{t^{k+1}} \right) (\mu^k - \mu^{k-1}) \]
\[ \text{dokler} \] pogoji niso izpolnjeni
while (jj < max_iter) && (cond < 0)) {
    jj++;
    copy_vec_part_negate(v, tmp_var_p, 40);
    mat_vec_mult_sparse(&CT, tmp_var_p, tmp_var_n);
    vec_sub(tmp_var_n, q1, tmp_var_n, 60);
    stack_vec(tmp_var_n, q2, rhs, 60, 40);
    perm_fwdsolve(&L, p, rhs, tmp_var_nm);
    mat_vec_mult_sparse(&Dinv, tmp_var_nm, tmp_var_nm2);
    backsolve_perm(&LT, p, tmp_var_nm2, tmp_var_nm);
    copy_vec_part(tmp_var_nm, x, 60);
    mat_vec_mult_sparse(&C, x, tmp_var_p);
    vec_add(v, tmp_var_p, tmp_var_p, 40);
    copy_vec_part(tmp_var_p, arg_prox_h, 40);
    clip_soft(tmp_var_p, l, u, (double *) &soft, 40);
    mat_vec_mult_diag(&Einv, tmp_var_p, y);
    copy_vec_part(lambda, lambda_old, 40);
    vec_sub(arg_prox_h, tmp_var_p, lambda, 40);
    vec_sub(lambda, lambda_old, tmp_var_p, 40);

    theta_old = theta;
    theta = (1+sqrt(1+pow(theta_old, 2)))/2;
    scalar_mult((theta_old - 1)/theta, tmp_var_p, 40);
    copy_vec_part(v, v_old, 40);
    vec_add(tmp_var_p, lambda, v, 40);

    if (mod(jj, 10) == 0) {
        cond = check_stop_cond_FGM(&Einvs, lambda, lambda_old, tmp_var_p, tmp_var_p2, 40, 1e-09);
    }
}

restart(lambda, lambda_old, v, v_old, tmp_var_p, tmp_var_p2, 40);
Restart

(B. O’Donoghue, E. Candès, Adaptive Restart for Accelerated Gradient Schemes, Found Comput Math 2013.)
• Upper bound on the required number of floating point operations that in turn stems from an upper bound on the iteration count
Complexity certification

- Upper bound on the required number of floating point operations that in turn stems from an upper bound on the iteration count
- accuracy => no. of iterations => no. of operations => computing time
• Explicit: fixed complexity
Complexity certification

• Explicit: fixed complexity
• Active set:
Complexity certification

• Explicit: fixed complexity

• Active set:
  – Finite termination
Complexity certification

• Explicit: fixed complexity
• Active set:
  – Finite termination
  – Worst case number of iterations huge
Complexity certification

- Explicit: fixed complexity
- Active set:
  - Finite termination
  - Worst case number of iterations huge
  - Early termination unexplored
Complexity certification

- Explicit: fixed complexity
- Active set:
  - Finite termination
  - Worst case number of iterations huge
  - Early termination unexplored
- Interior point:
Complexity certification

• Explicit: fixed complexity
• Active set:
  – Finite termination
  – Worst case number of iterations huge
  – Early termination unexplored
• Interior point:
  – Certificates very conservative
Complexity certification

• Explicit: fixed complexity

• Active set:
  - Finite termination
  - Worst case number of iterations huge
  - Early termination unexplored

• Interior point:
  - Certificates very conservative

• First order:
Complexity certification

- Explicit: fixed complexity
- Active set:
  - Finite termination
  - Worst case number of iterations huge
  - Early termination unexplored
- Interior point:
  - Certificates very conservative
- First order:
  - Certificates within few orders of magnitude
• State of the art for first order methods:
• State of the art for first order methods:
  – Derived for certain (simpler) algorithms
• State of the art for first order methods:
  - Derived for certain (simpler) algorithms
    (not generalized, no restarting)
• State of the art for first order methods:
  - Derived for certain (simpler) algorithms
    (not generalized, no restarting)
  - Derived for certain (simpler) constraints
• State of the art for first order methods:
  – Derived for certain (simpler) algorithms
    (not generalized, no restarting)
  – Derived for certain (simpler) constraints
• Convex optimization
• Convex optimization
• Does not mention duality
• MPC with some QP
• MPC with some QP
• Duality: partial Lagrange relaxation (simple state constraints), shows that complete is worse, but complete is what we need
• MPC with some QP
• Duality: partial Lagrange relaxation (simple state constraints), shows that complete is worse, but complete is what we need
• Duality: dual problem not strongly concave
• What to do? Transform the MPC-obtained QP into dual space
What to do? Transform the MPC-obtained QP into dual space

Follow either Richter or Nesterov with dual QP
• What to do? Transform the MPC-obtained QP into dual space
• Follow either Richter or Nesterov with dual QP
• Preconditioning?
• Restarting?