Operator Splitting Methods for Large-scale Convex Optimisation

Paul Goulart

Control Group, Department of Engineering Science, University of Oxford

www.eng.ox.ac.uk/control
Some collaborators

Bartolomeo Stellato
Goran Banjac
Yang Zheng
Michael Garstka
Nikitas Rontsis
Antonis Papachristodoulou

Stephen Boyd
Nick Moehle
Andy Wynn
Giovanni Fantuzzi
Example Applications
Problems: Sparse minimisation

The LASSO problem

\[
\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1
\]
Problems: Real-time optimal control

\[ V^*(x) = \min \sum_{k=0}^{N-1} \ell(x_k, u_k) \]

subject to \[ x_0 = x, \]
\[ x_{k+1} = f(x_k, u_k) \]
\[ u_k \in U \]
\[ x_k \in X \]
Problems: Flow stability

A Lyapunov function for a 9-D fluid model:

Solvable as a polynomial ‘sum-of-squares’ problem.
Operator Splitting Methods
Many optimisation problems can be written like this:

\[
\begin{align*}
\text{minimize} & \quad f(w) + g(w) \\
\text{subject to} & \quad \tilde{w} = w
\end{align*}
\]

\[
\begin{align*}
\min & \quad f(\tilde{w}) + g(w) + \frac{\rho}{2}\|w - \tilde{w}\|^2 \\
\text{subject to} & \quad \tilde{w} = w
\end{align*}
\]

\[
\begin{align*}
\max_y & \left[ \min_{(w,\tilde{w})} f(\tilde{w}) + g(w) + \frac{\rho}{2}\|w - \tilde{w}\|^2 + \langle y, w - \tilde{w} \rangle \right]
\end{align*}
\]
Solving the dual problem:

\[
\max_y \left[ \min_{(w, \tilde{w})} \left( f(\tilde{w}) + g(w) + \frac{\rho}{2} \| w - \tilde{w} \|^2 + \langle y, w - \tilde{w} \rangle \right) \right]
\]

Gradient ascent (difficult way):

\[
(w^{k+1}, \tilde{w}^{k+1}) \leftarrow \arg \min_{(w, \tilde{w})} \left( f(\tilde{w}) + g(w) + \frac{\rho}{2} \| \tilde{w} - w + \frac{y^k}{\rho} \|^2 \right)
\]

\[
y^{k+1} \leftarrow y^k + \rho (\tilde{w}^{k+1} - w^{k+1})
\]
Alternating Direction Method of Multipliers (ADMM)

Solving the dual problem:

\[
\max_y \left[ \min_{(w, \tilde{w})} f(\tilde{w}) + g(w) + \frac{\rho}{2} \| w - \tilde{w} \|^2 + \langle y, w - \tilde{w} \rangle \right]
\]

Gradient ascent (easy way / ADMM):

1. \( \tilde{w}^{k+1} \leftarrow \arg \min_{\tilde{w}} \left( f(\tilde{w}) + \frac{\rho}{2} \| \tilde{w} - w^k + \frac{y^k}{\rho} \|^2 \right) \)

2. \( w^{k+1} \leftarrow \arg \min_w \left( g(w) + \frac{\rho}{2} \| w - \tilde{w}^{k+1} - \frac{y^k}{\rho} \|^2 \right) \)

3. \( y^{k+1} \leftarrow y^k + \rho \left( \tilde{w}^{k+1} - w^{k+1} \right) \)
A standard form optimisation problem

All of my optimisation problems can be written like this:

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} x^\top P x + q^\top x \\
\text{subject to} \quad & Ax \in C
\end{align*}
\]

- **QP**: $C$ is a (translated) box or positive orthant.
- **SDP**: $C$ is the (translated) positive semidefinite cone.
A standard form optimisation problem

All of our optimisation problems can be written like this:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} x^\top P x + q^\top x \\
\text{subject to} & \quad Ax = z, \quad z \in \mathcal{C}
\end{align*}
\]

\[
\begin{align*}
\min & \quad \frac{1}{2} \tilde{x}^\top P \tilde{x} + q^\top \tilde{x} + \mathcal{I}_{A\tilde{x} = \tilde{z}}(\tilde{x}, \tilde{z}) + \mathcal{I}_{\mathcal{C}}(z) \\
\text{s.t.} & \quad (\tilde{x}, \tilde{z}) = (x, z) \\
& \quad \tilde{w} = w
\end{align*}
\]
Complete ADMM Algorithm

Conic Optimisation with ADMM:

1. \( (x^{k+1}, z^{k+1}) \leftarrow \arg\min_{(x, z): Ax = \bar{z}} \frac{1}{2} x^T P x + q^T x + \frac{\sigma}{2} \| x - x^k \|^2 + \frac{\rho}{2} \| \tilde{z} - z^k + \frac{y^k}{\rho} \|^2 \)

2. \( z^{k+1} \leftarrow \Pi_C \left( \tilde{z}^{k+1} + \frac{y^k}{\rho} \right) \)

3. \( y^{k+1} \leftarrow y^k + \rho \left( \tilde{z}^{k+1} - z^{k+1} \right) \)
Solving the inner QP

\[ \text{minimize} \quad \frac{1}{2} x^T Px + q^T x + \frac{\sigma}{2} \left\| x - x^k \right\|^2 + \frac{\rho}{2} \left\| z - z^k + \frac{y^k}{\rho} \right\|^2 \]

s.t. \quad Ax = z

**Direct Method**

Solve the KKT system:

\[
\begin{bmatrix}
P + \sigma I & A^T \\
A & -\frac{1}{\rho} I
\end{bmatrix}
\begin{bmatrix}
x \\
\nu
\end{bmatrix}
=
\begin{bmatrix}
\sigma x^k - q \\
z^k - \frac{1}{\rho} y^k
\end{bmatrix}
\]

**Quasi-definite matrix**

**Always solvable**

**LDL^T factorable**

**Factorization caching**
Solving the inner QP

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2}x^T Px + q^T x + \frac{\sigma}{2} \left\| x - x^k \right\|^2 + \frac{\rho}{2} \left\| z - z^k + \frac{y^k}{\rho} \right\|^2 \\
\text{s.t.} & \quad Ax = z
\end{align*}
\]

**Indirect Method**

Solve the system:

\[
(P + \sigma I + \rho A^T A) x = \sigma x^k - q + A^T (\rho z^k - y^k)
\]

*Positive definite matrix*

Conjugate Gradient

A and P can be huge
Complete ADMM Algorithm

Implementation (easy version / ADMM):

1. \((x^{k+1}, \tilde{z}^{k+1}) \leftarrow \arg \min_{(x,z): Ax = z} \frac{1}{2} x^T P x + q^T x + \frac{\sigma}{2} \|x - x^k\|^2 + \frac{\rho}{2} \|z - \tilde{z}^k + \frac{y^k}{\rho}\|^2\)

2. \(z^{k+1} \leftarrow \Pi_C \left(\tilde{z}^{k+1} + \frac{y^k}{\rho}\right)\)

3. \(y^{k+1} \leftarrow y^k + \rho \left(\tilde{z}^{k+1} - z^{k+1}\right)\)
Computing the projection $\Pi_C(v)$:

- **QP**: $C$ is a box or positive orthant.
- **SDP**: $C$ is the positive semidefinite cone.
Computing the projection $\Pi_C(v)$:

- **QP**: $C$ is a box or positive orthant.
- **SDP**: $C$ is the positive semidefinite cone.

$\Pi_C(v) = \max(\min(v, u), l)$

Projection scales linearly with problem size.
Computing the projection $\Pi_C(v)$:

- **QP**: $C$ is a box or positive orthant.
- **SDP**: $C$ is the positive semidefinite cone.

$$ v = U\Lambda U^\top $$

$$ \Pi_C(v) = U \max(0, \Lambda) U^\top $$

Projection cost scales cubically with matrix size.
Chordal Decomposition

Chordal Graphs and Sparse Matrices

Set of nodes: \( V \)
Set of edges: \( E \subseteq V \times V \)
Graph : \( G = (V, E) \)

A chordal graph can be decomposed into its maximal cliques \( C = \{C_1, \ldots, C_n\} \)
Matrices defined on a graph

\( \mathbb{S}^n(\mathcal{E}, 0) = \) Set of \( n \times n \) symmetric matrices with zeros outside of \( \mathcal{E} \)

\( \mathbb{S}^n_+(\mathcal{E}, 0) = \{ X \in \mathbb{S}^n(\mathcal{E}, 0) \mid X \succeq 0 \} \).
A decomposition theorem

Given a chordal graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with maximal cliques $\{C_1, C_2, \ldots, C_p\}$

**Aglar’s theorem**

$A \in S^n_+(\mathcal{E}, 0)$ if and only if there exists matrices $M_k \in S^n_+(C_k)$ such that

$$A = \sum_{k=1}^{p} M_k$$
A decomposition theorem

Given a chordal graph $G(V, E)$ with maximal cliques $\{C_1, C_2, \ldots, C_p\}$

**Aglar’s theorem**

$A \in S^n_+(E, 0)$ if and only if there exist matrices $M_k \in S^n_+(C_k)$ such that $A = \sum_{k=1}^{p} M_k$
Every semidefinite program can be written like this:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} x^\top P x + q^\top x \\
\text{subject to} & \quad A_0 + \sum_{i=1}^m x_i A_i \succeq 0
\end{align*}
\]

- Aggregate sparsity pattern \( \mathcal{E} \) from \( A_0, A_1, \ldots, A_m \)
# Create OSQP object
m = osqp.OSQP()

# Initialize solver
m.setup(P, q, A, l, u, settings)

# Generate C code
m.codegen('folder_name')
Performance Profiles (QP Solver)

The diagram illustrates the performance of different QP solvers, including OSQP, GUROBI, ECOS, MOSEK, and qpOASES, measured in terms of the ratio of problems solved against the performance ratio. The y-axis represents the ratio of problems solved, while the x-axis shows the performance ratio. Each solver is represented by a different line color or marker, allowing for a comparison of their performance across different problem sizes.

- **OSQP**: Typically performs well, solving a high ratio of problems even at lower performance ratios.
- **GUROBI**: Generally maintains a high ratio throughout, indicating consistent performance.
- **ECOS**: Shows a steady increase in the ratio of problems solved, suggesting gradual improvement.
- **MOSEK**: Exhibits a strong performance, solving a high ratio of problems at various performance ratios.
- **qpOASES**: Demonstrates a slower but steady increase in the ratio of problems solved over the performance ratio.

The graph provides a visual representation of how each solver handles different problem sizes, aiding in the selection of the most suitable solver for specific applications.
Scaling Performance (SDPs)

![Graph showing performance scaling for different algorithms with varying numbers of constraints and block sizes.]

- **SeDuMi**
- **SparseCoLo+SeDuMi**
- **SCS**
- **SparseCoLo+SCS**
- **CDCS (primal)**
- **CDCS (dual)**

**Parameters:****
- Number of constraints, \( m \)
- Size of each block, \( d \)
- Number of blocks, \( l \)

**Time (s)轴**: Logarithmic scale from 10^0 to 10^4.

**Size of each block, \( d \)轴**: 10 to 200.

**Number of blocks, \( l \)轴**: 10 to 40.

**Key**:
- □ SeDuMi
- + SparseCoLo+SeDuMi
- ○ SCS
- △ SparseCoLo+SCS
- ♦ CDCS (primal)
- ▼ CDCS (dual)
Conclusion

QPs and SDPs can be solved very fast.

QPs and SDPs can be solved even when really huge.

QPs and (soon...) SDPs work in real time on embedded systems.

Everything is open source: https://github.com/oxfordcontrol