Totally Asynchronous Distributed Quadratic Programming

Matthew Hale

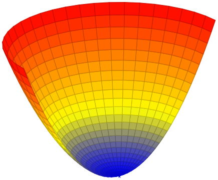
Department of Mechanical and Aerospace Engineering
University of Florida

AFOSR Center of Excellence on Assured Autonomy in Contested Environments
October 15, 2019

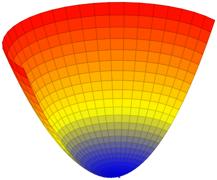
QPs Arise Across Control and Optimization

- QPs take the general form

$$\underset{x}{\text{minimize}}\ \frac{1}{2}x^TQx + r^Tx$$



QPs Arise Across Control and Optimization

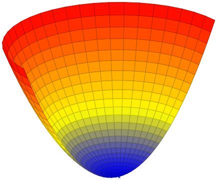


- QPs take the general form

$$\underset{x}{\text{minimize}}\ \frac{1}{2}x^T Q x + r^T x$$

- Appear explicitly in
  - Quadrotor trajectory generation
  - Numerical optimal control
  - Statistical learning

QPs Arise Across Control and Optimization



- QPs take the general form

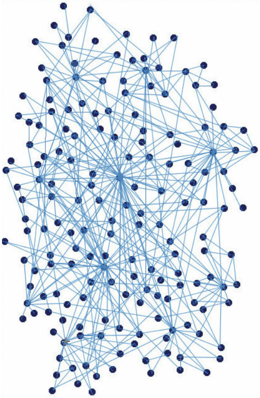
$$\underset{x}{\text{minimize}}\ \frac{1}{2}x^TQx + r^Tx$$

- Appear explicitly in
  - Quadrotor trajectory generation
  - Numerical optimal control
  - Statistical learning

Overall Motivation

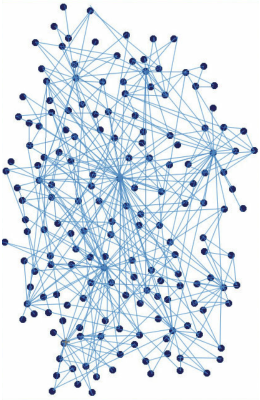
Design a multi-agent framework for solving QPs.

- Adversaries can disrupt agents' communications



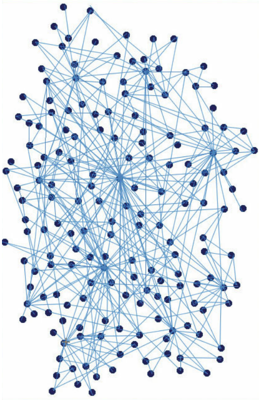
Solving QPs in Contested Environments

- Adversaries can disrupt agents' communications
  - We don't want agents to wait and synchronize between computations
  - We don't want to require bounded delays



Solving QPs in Contested Environments

- Adversaries can disrupt agents' communications
  - We don't want agents to wait and synchronize between computations
  - We don't want to require bounded delays



Algorithmic Goal

We want to allow *totally* asynchronous operations by agents.

Distributing Computations Across Agents

- Agent i stores a local copy of all decision variables, denoted x^i

i $\quad x^i \neq x^j \quad$ j

Distributing Computations Across Agents

- Agent i stores a local copy of all decision variables, denoted x^i

$$i \qquad x^i \neq x^j \qquad j$$

- Want agents to update only a small subset of system variables
  - Promotes scalability
  - Amenable to control problems in which agents compute trajectories/control decisions

Distributing Computations Across Agents

- Agent i stores a local copy of all decision variables, denoted x^i

$$\textcircled{i} \quad x^i \neq x^j \quad \textcircled{j}$$

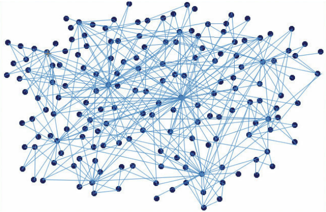
- Want agents to update only a small subset of system variables
  - Promotes scalability
  - Amenable to control problems in which agents compute trajectories/control decisions

- Agent i updates only $x^i_{[i]}$
- Agent i waits to receive $x^i_{[j]}$ from agent j

$$x^i = \begin{pmatrix} x^i_{[1]} \\ \vdots \\ x^i_{[i]} \\ \vdots \\ x^i_{[n]} \end{pmatrix}$$

Gradients are Robust to Asynchrony

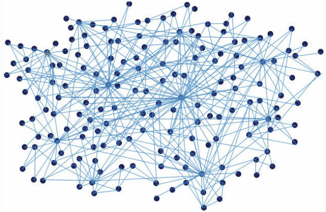
- Asynchrony requires a sufficiently robust update law
- It should also be simple to decentralize and scale up



UF UNIVERSITY of FLORIDA Duke UNIVERSITY TEXAS The University of Texas at Austin UC SANTA CRUZ

Gradients are Robust to Asynchrony

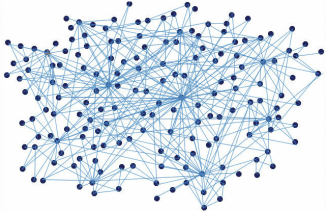
- Asynchrony requires a sufficiently robust update law
- It should also be simple to decentralize and scale up



- Further want agents to execute updates as fast as possible

Gradients are Robust to Asynchrony

- Asynchrony requires a sufficiently robust update law
- It should also be simple to decentralize and scale up



- Further want agents to execute updates as fast as possible
- We will use gradient descent as the template update law:

$$\begin{aligned} x(k+1) &= x(k) - \gamma \nabla f\big(x(k)\big) \\ &= x(k) - \gamma\big(Qx(k) + r\big) \end{aligned}$$

Proposed Distributed QP Algorithm

- Split up Q and r via

$$Q = \begin{pmatrix} Q^{[1]} \\ Q^{[2]} \\ \vdots \\ Q^{[N]} \end{pmatrix} \qquad r = \begin{pmatrix} r^{[1]} \\ r^{[2]} \\ \vdots \\ r^{[N]} \end{pmatrix}$$

Proposed Distributed QP Algorithm

- Split up Q and r via

$$Q = \begin{pmatrix} Q^{[1]} \\ Q^{[2]} \\ \vdots \\ Q^{[N]} \end{pmatrix} \qquad r = \begin{pmatrix} r^{[1]} \\ r^{[2]} \\ \vdots \\ r^{[N]} \end{pmatrix}$$

Algorithm 1

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \gamma\left(Q^{[i]}x^i(k) + r^{[i]}\right) & \text{agent } i \text{ updates at time } k \\ \end{cases}$$

Proposed Distributed QP Algorithm

- Split up Q and r via

$$Q = \begin{pmatrix} Q^{[1]} \\ Q^{[2]} \\ \vdots \\ Q^{[N]} \end{pmatrix} \qquad r = \begin{pmatrix} r^{[1]} \\ r^{[2]} \\ \vdots \\ r^{[N]} \end{pmatrix}$$

Algorithm 1

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \gamma\left(Q^{[i]} x^i(k) + r^{[i]}\right) & \text{agent } i \text{ updates at time } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

Proposed Distributed QP Algorithm

- Split up Q and r via

$$Q = \begin{pmatrix} Q^{[1]} \\ Q^{[2]} \\ \vdots \\ Q^{[N]} \end{pmatrix} \qquad r = \begin{pmatrix} r^{[1]} \\ r^{[2]} \\ \vdots \\ r^{[N]} \end{pmatrix}$$

Algorithm 1

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \gamma\left(Q^{[i]}x^i(k) + r^{[i]}\right) & \text{agent } i \text{ updates at time } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

$$x^i_{[j]}(k+1) = \begin{cases} x^j_{[j]} & i \text{ receives } j\text{'s state at time } k \end{cases}$$

Proposed Distributed QP Algorithm

- Split up Q and r via

$$Q = \begin{pmatrix} Q^{[1]} \\ Q^{[2]} \\ \vdots \\ Q^{[N]} \end{pmatrix} \qquad r = \begin{pmatrix} r^{[1]} \\ r^{[2]} \\ \vdots \\ r^{[N]} \end{pmatrix}$$

Algorithm 1

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \gamma\left(Q^{[i]}x^i(k) + r^{[i]}\right) & \text{agent } i \text{ updates at time } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

$$x^i_{[j]}(k+1) = \begin{cases} x^j_{[j]} & i \text{ receives } j\text{'s state at time } k \\ x^i_{[j]}(k) & \text{otherwise} \end{cases}$$

Convergence of Distributed QP Algorithm

Theorem 1: Convergence

Suppose that

- Q is diagonally dominant
- $\gamma < \frac{1}{Q_{ii}}$ for all i.

Convergence of Distributed QP Algorithm

Theorem 1: Convergence

Suppose that

- Q is diagonally dominant
- $\gamma < \frac{1}{Q_{ii}}$ for all i.

Then $\|x^i(k) - \hat{x}\|_2 \to 0$ for all i and, for $q \in (0,1)$,

$$\underbrace{\max_{i \in [N]} \|x^i(k) - \hat{x}\|}_{V(x(k))} \leq q^{\mathsf{ops}(k)} \underbrace{\max_{i \in [N]} \|x^i(0) - \hat{x}\|}_{V(x(0))}$$

Convergence of Distributed QP Algorithm

Theorem 1: Convergence

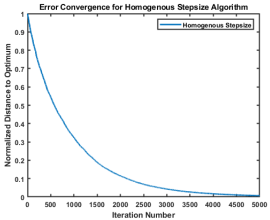
Suppose that

- Q is diagonally dominant
- $\gamma < \frac{1}{Q_{ii}}$ for all i.

Then $\|x^i(k) - \hat{x}\|_2 \to 0$ for all i and, for $q \in (0,1)$,

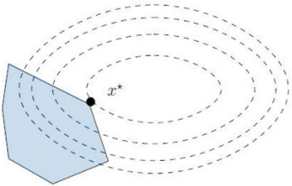
$$\underbrace{\max_{i\in[N]} \|x^i(k) - \hat{x}\|}_{V(x(k))} \leq q^{\mathsf{ops}(k)} \underbrace{\max_{i\in[N]} \|x^i(0) - \hat{x}\|}_{V(x(0))}$$

- Convergence is (imperfectly) geometric



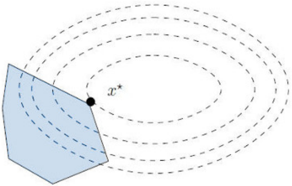
Convergence Can Be Slow

- Lots of "reasonable" QPs are not well-conditioned
- Formally, if $k_Q = \frac{\lambda_1(Q)}{\lambda_n(Q)}$ is large, then convergence is slow



Convergence Can Be Slow

- Lots of "reasonable" QPs are not well-conditioned
- Formally, if $k_Q = \frac{\lambda_1(Q)}{\lambda_n(Q)}$ is large, then convergence is slow

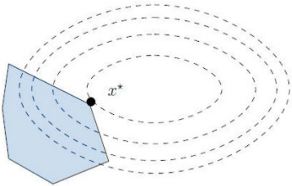


- Convergence rate is dictated by $q \in (0,1)$ here, which is

$$q := \sup_{\|x\|_\infty = 1} \max_{k \in [N]} \left\| \left(I^{[k]} - \gamma Q^{[k]} \right) x \right\|_2$$

Convergence Can Be Slow

- Lots of "reasonable" QPs are not well-conditioned
- Formally, if $k_Q = \frac{\lambda_1(Q)}{\lambda_n(Q)}$ is large, then convergence is slow



- Convergence rate is dictated by $q \in (0, 1)$ here, which is

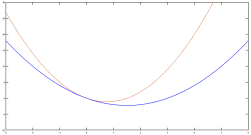
$$q := \sup_{\|x\|_\infty = 1} \max_{k \in [N]} \left\| \left(I^{[k]} - \gamma Q^{[k]} \right) x \right\|_2$$

- As $k_Q \to \infty$, find $q \to 1$ and convergence comes to a halt

Heterogeneous Parameter Selection

- We can regularize to make them better: $Q + A$ replaces Q, now solve

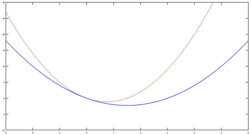
$$\underset{x \in X}{\text{minimize}} \ \frac{1}{2} x^T (Q + A) x + r^T x$$



Heterogeneous Parameter Selection

- We can regularize to make them better: $Q + A$ replaces Q, now solve

$$\underset{x \in X}{\text{minimize}} \ \frac{1}{2} x^T (Q + A) x + r^T x$$

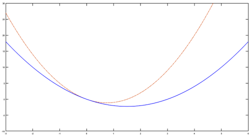


- Conventionally, $A = \alpha I$, but this requires agreeing on α

Heterogeneous Parameter Selection

- We can regularize to make them better: $Q + A$ replaces Q, now solve

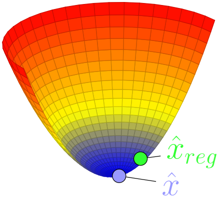
$$\underset{x \in X}{\text{minimize}} \ \frac{1}{2} x^T (Q + A) x + r^T x$$



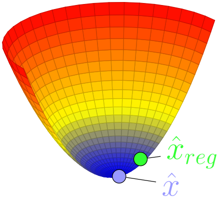
- Conventionally, $A = \alpha I$, but this requires agreeing on α
- We'd rather not have to agree on γ either

Independently Regularizing

- Regularizing Q changes the solution
- Small regularizations $\Rightarrow$ small error

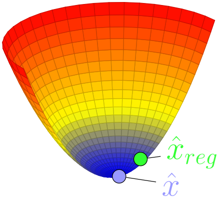


- Regularizing Q changes the solution
- Small regularizations $\Rightarrow$ small error
- Consider relative error $e(A) = \frac{|f(\hat{x}_{reg}) - f(\hat{x})|}{|f(\hat{x})|}$



Independently Regularizing

- Regularizing Q changes the solution
- Small regularizations $\Rightarrow$ small error
- Consider relative error $e(A) = \frac{|f(\hat{x}_{reg}) - f(\hat{x})|}{|f(\hat{x})|}$



Theorem: Regularization Error

For $\epsilon \in (0,1)$, get $e(A) \leq \epsilon$ if

$$\alpha_i \leq \frac{\sqrt{\epsilon}}{1-\sqrt{\epsilon}} \underbrace{\left(\left|Q_{ii}^{[i]}\right| - \sum_{\substack{j=1 \\ j \neq i}}^{n} \left|Q_{ij}^{[i]}\right| \right)}_{\text{How diagonally dominant row } i \text{ is}}$$

- Only requires knowledge of $Q^{[i]}$!

Independently Choosing Stepsizes

- Want stepsize rules that also depend only upon $Q^{[i]}$ and $r^{[i]}$

$$\gamma_i := \gamma_i\left(\boxed{\quad Q^{[i]} \quad}\,,\,\boxed{r^{[i]}}\right)$$

Independently Choosing Stepsizes

- Want stepsize rules that also depend only upon $Q^{[i]}$ and $r^{[i]}$

$$\gamma_i := \gamma_i\left(\boxed{\quad Q^{[i]} \quad} , \boxed{r^{[i]}} \right)$$

- No reason for agent i to use one stepsize for every variable

Independently Choosing Stepsizes

- Want stepsize rules that also depend only upon $Q^{[i]}$ and $r^{[i]}$

$$\gamma_i := \gamma_i \left(\boxed{\quad Q^{[i]} \quad} \,,\, \boxed{r^{[i]}} \right)$$

- No reason for agent i to use one stepsize for every variable
- For ℓ^{th} variable, choose

$$\gamma_\ell < \frac{2}{\sum_{j=1}^{N} \left| Q_{kj} \right| + \alpha_\ell}$$

Algorithm 2

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \Gamma_i\left(Q^{[i]}x^i(k) + r^{[i]} + A_i x^i_{[i]}(k)\right) & i \text{ updates at } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

Algorithm 2

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \Gamma_i\left(Q^{[i]}x^i(k) + r^{[i]} + A_i x^i_{[i]}(k)\right) & i \text{ updates at } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

$$x^i_{[j]}(k+1) = \begin{cases} x^j_{[j]} & i \text{ receives } j\text{'s state at time } k \\ x^i_{[j]}(k) & \text{otherwise} \end{cases}$$

Algorithm 2

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \Gamma_i \big(Q^{[i]} x^i(k) + r^{[i]} + A_i x^i_{[i]}(k) \big) & i \text{ updates at } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

$$x^i_{[j]}(k+1) = \begin{cases} x^j_{[j]} & i \text{ receives } j\text{'s state at time } k \\ x^i_{[j]}(k) & \text{otherwise} \end{cases}$$

- Converges geometrically again:

$$\max_{i \in [N]} \|x^i(k) - \hat{x}\| \leq q_{reg}^{\mathsf{ops}(k)} \max_{i \in [N]} \|x^i(0) - \hat{x}\|$$

Algorithm 2

For all i and all k:

$$x^i_{[i]}(k+1) = \begin{cases} x^i_{[i]}(k) - \Gamma_i\big(Q^{[i]}x^i(k) + r^{[i]} + A_i x^i_{[i]}(k)\big) & i \text{ updates at } k \\ x^i_{[i]}(k) & \text{otherwise} \end{cases}$$

$$x^i_{[j]}(k+1) = \begin{cases} x^j_{[j]} & i \text{ receives } j\text{'s state at time } k \\ x^i_{[j]}(k) & \text{otherwise} \end{cases}$$

- Converges geometrically again:

$$\max_{i\in[N]} \|x^i(k) - \hat{x}\| \leq q_{reg}^{\text{ops}(k)} \max_{i\in[N]} \|x^i(0) - \hat{x}\| < q^{\text{ops}(k)} \max_{i\in[N]} \|x^i(0) - \hat{x}\|$$

- Solving

$$\underset{x\in X}{\text{minimize}}\ \frac{1}{2}x^TQx + r^Tx$$

- Have $k_Q = 100$
- Want $e(A) \leq 0.05$

Numerical Convergence

- Solving

$$\underset{x \in X}{\text{minimize}} \; \frac{1}{2} x^T Q x + r^T x$$

- Have $k_Q = 100$
- Want $e(A) \leq 0.05$
- Then $\alpha_i \leq 0.29$, use stepsize rule
- $k_Q = 77.8$ now