# ALADIN-α – An open-source MATLAB toolbox for distributed non-convex optimization

**Preprint** · June 2020

CITATIONS
0

READS
497

**6 authors**, including:

Alexander Engelmann
Technische Universität Dortmund
**38** PUBLICATIONS **209** CITATIONS
SEE PROFILE

Yuning Jiang
École Polytechnique Fédérale de Lausanne
**53** PUBLICATIONS **233** CITATIONS
SEE PROFILE

Ruchuan Ou
Technische Universität Dortmund
**5** PUBLICATIONS **9** CITATIONS
SEE PROFILE

Boris Houska
ShanghaiTech University
**139** PUBLICATIONS **3,339** CITATIONS
SEE PROFILE

# ALADIN-$\alpha$ – An open-source MATLAB toolbox for distributed non-convex optimization

Alexander Engelmann*[1], Yuning Jiang[2], Henrieke Benner[1], Ruchuan Ou[1], Boris Houska[2], and Timm Faulwasser[1,3]

[1] *Institute for Automation and Applied Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[2] *School of Information Science and Technology, ShanghaiTech University, Shanghai, China*
[3] *currently with the Institute of Energy Systems, Energy Efficiency and Energy Economics, TU Dortmund University, Dortmund, Germany*

June 4, 2020

This paper introduces an open-source software for *distributed* and *decentralized* non-convex optimization named ALADIN-$\alpha$. ALADIN-$\alpha$ is a MATLAB implementation of the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) algorithm, which is tailored towards rapid prototyping for non-convex distributed optimization. An improved version of the recently proposed bi-level variant of ALADIN is included enabling *decentralized* non-convex optimization. A collection of application examples from different applications fields including chemical engineering, robotics, and power systems underpins the application potential of ALADIN-$\alpha$.

***Index terms***— Distributed Optimization, Decentralized Optimization, Nonconvex Optimization, ALADIN, ADMM, Optimal Power Flow, Distributed Model Predictive Control

## 1. Introduction

Distributed *non-convex* optimization is of significant interest in various engineering domains. These domains range from electrical power systems [1]–[4], via transportation

[0] **Abbreviations:** ALADIN, Augmented Lagrangian Alternating Direction Inexact Newton; ADMM, Alternating Direction Method of Multipliers; QP, Quadratic Program; NLP, Nonlinear Program; LICQ, Linear Independence Constraint Qualification; OCP, Optimal Control Problem; CG, Conjugate Gradients

problems [5], to machine learning [6], as well as distributed control [5], [7]–[9], and distributed estimation [10]–[13]. Surprisingly, only very few software toolboxes for distributed optimization are available. Moreover, even if one can find one of the rare examples, these tools are typically tailored to a specific application at hand and typically address *convex* problems. Examples are several versions of the Alternating Direction of Multipliers Method (ADMM) applied to a plethora of applications summarized in Boyd et al. [6], with code available in [1]. An implementation of ADMM for consensus problems can be found in [2]. A tailored implementation of ADMM for Optimal Power Flow problems using an algorithm from Guo et al. [14] can be found under [3]. However, there is a lack of multi-purpose software tools for distributed optimization and we were not able to find any open-source implementations for generic distributed non-convex optimization problems. Also with respect to *decentralized* non-convex optimization, we were not able to find any publicly available code.

Notice that we distinguish *parallel* and *distributed* optimization. In *parallel* optimization, the main motivation is to *speed-up* computation or to attain computational tractability in the first place, i.e. communication and the amount of central coordination of secondary importance (shared memory architectures). In contrast, in *distributed* optimization, a main goal is to minimize central coordination and communication as much as possible (distributed memory architectures). *Decentralized* optimization additionally requires communication purely on a neighbor-to-neighbor basis. This is especially relevant in multi-agent settings, where individual entities cooperate to the end of optimization, control, or estimation (e.g. in the context of cyber-physical systems, IoT, or embedded control).

For *parallel optimization* efficient structure-exploiting tools exist. A closed-source parallel interior point software is `OOPS` [15]. The open-source package `qpDunes` is tailored towards parallel solutions of Quadratic Programs (QPs) arising in model predictive control [16]. For general QPs, the partially parallelizable solver `OSQP` seems promising [17]. `PIPS` is a collection of algorithms solving structured linear programs, QPs, and general Nonlinear Programming Problems (NLPs) in parallel [18], [19]. The software `HiOp` is tailored towards structured and very large-scale NLPs with few nonlinear constraints based on interior point methods[20], [21]. Moreover, combining parallel linear algebra routines (e.g. `PARDISO` [22]) with standard nonlinear programming solvers (e.g. `IPOPT` [23]) also leads to partially parallel algorithms [24], [25]. All these tools are implemented in low-level languages such as `C` or `C++` leading to a high computational performance. On the other hand, their focus is mainly *computational speedup* via parallel computing rather than distributed and decentralized optimization in a multi-agent setting.

Considering a multi-agent setting and problems with non-convexities, classical distributed algorithms based on Lagrangian relaxation such as dual decomposition or ADMM are guaranteed to converge only for specific non-convexities [26]–[28] commonly at a sublinear/linear rate [29], [30]. One of the few algorithms exhibiting convergence

[1] `https://web.stanford.edu/~boyd/papers/admm/`
[2] `http://users.isr.ist.utl.pt/~jmota/DADMM/`
[3] `https://github.com/guojunyao419/OPF-ADMM`

guarantees at a *superlinear* or even *quadratic* rate for problems with differentiable non-convexities is the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) algorithm. Yet—up to now—also for ALADIN a publicly available software implementation seems to be missing. The present paper takes first steps in this direction introducing an open-source MATLAB implementation named ALADIN-$\alpha$. ALADIN-$\alpha$ is intended for rapid algorithmic prototyping and aims at user-friendliness. The only user-provided information are objective and constraint functions—derivatives and numerical solvers are generated automatically by algorithmic differentiation routines and external state-of-the-art NLP solvers. A rich set of examples coming with ALADIN-$\alpha$ covers problems from robotics, power systems, sensor networks and chemical engineering underpinnning the application potential of ALADIN as such and of ALADIN-$\alpha$. Besides the vanilla ALADIN algorithm [31], ALADIN-$\alpha$ covers recent algorithmic extensions including:

- the bi-level ALADIN extension with decentralized variants of the Conjugate Gradient (CG) method and the Alternating Direction of Multipliers Method (ADMM) as inner algorithms allowing for *decentralized non-convex optimization* [1];
- BFGS Hessian functionalities offering a means to *reduce communication and computation* [32];
- the nullspace ALADIN variant which *reduces communication and coordination* [1];
- consideration of NLPs with parametric data enabling *distributed Model Predictive Control (MPC)*, and
- heuristics for Hessian regularization and parameter tuning for *improving performance.*

Moreover, we provide an implementation of ADMM based on the formulation of Houska et al. [31] which uses the same interface as ALADIN. This way, comparisons between ALADIN and ADMM are fostered. Notice, ALADIN-$\alpha$ can be executed *in parallel mode* via the MATLAB parallel computing toolbox. This often leads to a substantial speed-up, for example, in distributed estimation problems. A documentation and many application examples of ALADIN-$\alpha$ are available under `https://alexe15.github.io/ALADIN.m/`. We remark that the primary focus of ALADIN-$\alpha$ is to provide a *rapid-prototyping* environment supporting algorithmic testing of distributed and decentralized algorithms for non-convex optimization. At this stage, computational speed or real-time feasibility is beyond the scope of the toolbox.

The remainder of the paper is organized as follows: Section 2 recalls the main ideas of ALADIN and bi-level ALADIN. In Subsection 3.2 we comment on the code structure and data structures. Section 4 presents a simple tutorial example illustrating the use of ALADIN-$\alpha$. Morover, a second example shows how to use ALADIN-$\alpha$ for distributed optimal control. In the Appendix we document implementation details.

# 2. Background on Distributed Optimization with ALADIN

## 2.1. Problem Formulation

ALADIN-$\alpha$ solves structured optimization problems of the form

$$\begin{aligned}
\min_{x_1,\dots,x_{n_s}} \quad & \sum_{i\in\mathcal{S}} f_i(x_i,p_i) && && && \text{(1a)}\\
\text{subject to} \quad & g_i(x_i,p_i)=0 && \mid \kappa_i, && \forall i\in\mathcal{S}, && \text{(1b)}\\
& h_i(x_i,p_i)\le 0 && \mid \gamma_i, && \forall i\in\mathcal{S}, && \text{(1c)}\\
& \underline{x}_i \le x_i \le \overline{x}_i && \mid \eta_i, && \forall i\in\mathcal{S}, && \text{(1d)}\\
& \sum_{i\in\mathcal{S}} A_i x_i = b && \mid \lambda, && && \text{(1e)}
\end{aligned}$$

where $\mathcal{S}=\{1,\dots,n_s\}$ is a set of subproblems, $f_i:\mathbb{R}^{n_{xi}}\times\mathbb{R}^{n_{pi}}\to\mathbb{R}$ are objective functions, $g_i:\mathbb{R}^{n_{xi}}\times\mathbb{R}^{n_{pi}}\to\mathbb{R}^{n_{gi}}$ and $h_i:\mathbb{R}^{n_{xi}}\times\mathbb{R}^{n_{pi}}\to\mathbb{R}^{n_{hi}}$ are the constraint functions for all subproblems $i\in\mathcal{S}$. The matrices $A_i\in\mathbb{R}^{n_c\times n_{xi}}$ contain affine coupling information between the subproblems. Lagrange multipliers $\kappa$ assigned to a constraint $g$ are denoted by $g(x)=0 \quad \mid \kappa$. The partially separable formulation of (1) is rather generic: it contains several other problem formulations as special cases such as consensus or sharing problems [6]. Note that problem (1) allows for parameter vectors $p_i\in\mathbb{R}^{n_{pi}}$: this can be useful for example in Model Predictive Control or if one would like to solve the same distributed problem multiple times for a variety of parameters. Problem (1) looks quite specialized at first glance. However, notice that most NLPs can be reformulated in form of (1) by introducing auxiliary variables. We discuss a concrete example for problem reformulation in Section 4.

## 2.2. Standard and bi-level ALADIN

ALADIN solves convex and non-convex optimization problems in form of (1) in a distributed fashion. A simplified flow chart of standard ALADIN is sketched in Figure 1. Rougly speaking, ALADIN can be described as a mix between ADMM and a Sequential Quadratic Programming (SQP) combining distribution computation from ADMM with fast convergence properties and guarantees from SQP [31]. Similar to ADMM, ALADIN adopts a parallel step—i.e., several NLPs are solved locally and in parallel minimizing local objective functions $f_i$ (plus augmentation terms $a_i$ accounting for coupling to other subsystems) subject to local constraints $g_i$ and $h_i$, cf. Subsection A.1. Since these NLPs are very similar in ALADIN and ADMM, both algorithms share the same computational complexity in this step. Sensitivities such as gradients, Hessians and Jacobian matrices of the local problems are evaluated locally in the next step of ALADIN. These sensitivities are collected in a coordination QP adopted from SQP methods. Note that the coordination QP is equality-constrained and strongly convex (under certain regularity assumptions)—thus it can be reformulated as a linear system of equations. The primal and dual solution vectors of this coordination QP iare broadcasted to the local subproblems and the next ALADIN iteration starts.

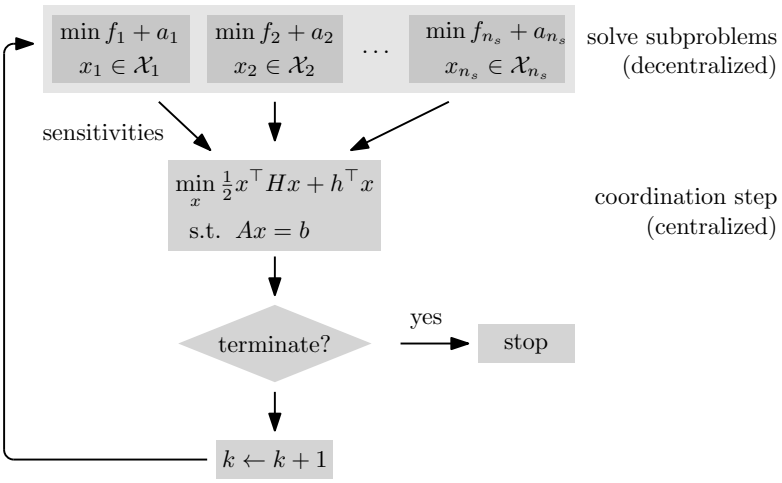



Figure 1: Simplified flow chart of standard ALADIN.

The main advantage of standard ALADIN over other existing approaches are convergence guarantees and fast local convergence. On the other hand, the coordination QP makes ALADIN distributed but *not decentralized*. Furthermore, the coordination step in standard ALADIN is quite heavy and communication intense compared with other approaches such as ADMM. Bi-level ALADIN tries to overcome these drawbacks by constructing a coordination QP of smaller dimension lowering communication. This lower-dimensional QP is solved in a decentralized fashion leading to an overall decentralized algorithm. A simplified flow chart of bi-level ALADIN is shown in Figure 2. Observe that in contrast to standard ALADIN (Figure 1), bi-level ALADIN tackles the coordination QP *in decentralized fashion* based on *neighbor-to-neighbor* communication. ALADIN-$\alpha$ comes with two decentralized algorithms: a decentralized version of the Conjugate Gradient (CG) method and a decentralized version of ADMM. Although these decentralized algorithms deliver only a certain level of precision, bi-level ALADIN is guaranteed to converge if certain error bounds hold [1]. For the sake of completeness, implementation details of the implemented ALADIN and bi-level ALADIN versions are given in the Appendix A.1.

## 2.3. Convergence Guarantees

Standard ALADIN—and also bi-level ALADIN—exhibit convergence guarantees. The set of assumptions required for ALADIN to converge are standard regularity conditions such as the Linear Independence Constraint Qualification (LICQ) and a slightly stronger form of the second-order sufficient condition [1].[4] If these conditions hold, if the coordination step is computed exactly, and if certain technical conditions hold, then ALADIN is guaranteed to converge locally at a superlinear/quadratic rate [31]. If the coordination step is solved inexactly, but the "inexactness" becomes increasingly smaller

[4]Note that these conditions are standard in nonlinear programming and used in many contexts [33].

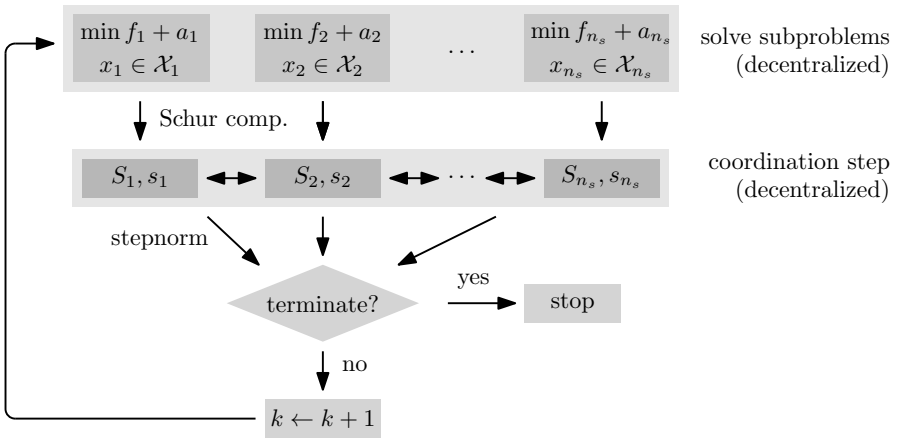



Figure 2: Simplified flow chart of bi-level ALADIN.

when approaching a local minimizer, these guarantees are preserved [1]. Similarly, if the local NLPs in the parallel step are solved inexactly, but the accuracy becomes increasingly higher when approaching a local minimizer, ALADIN's convergence guarantees are likewise preserved [32].

# 3. ALADIN-$\alpha$

## 3.1. Code Structure

In order to avoid side-effects and to make code-modification easy for beginners, we chose a procedual/functional programming style. We decided to implement all core features in MATLAB to enable easy rapid-prototyping. The overall structure of `run_ALADIN()`—which is the main function of ALADIN-$\alpha$—is shown in Figure 3. First, a reprocessing step performs a consistency check of the input data and provides default options. The `createLocSolAndSens()` function initializes the local parameterized NLPs and sensitivities for all subproblems $i \in \mathcal{S}$. For constructing the local NLPs and sensitivities, we use `CasADi` [34] due to its algorithmic differentiation features and the possibility to interface many state-of-the-art NLP solvers such as, for example, `IPOPT` [23]. `CasADi` itself relies on pre-compiled code making function and derivative evaluation fast. A `reuse` option allows to avoid the reconstruction of the `CasADi` problem setup in case of saved problem formulations. When the `reuse` mode is activated (e.g. when ALADIN-$\alpha$ is used within an MPC loop), `createLocSolAndSens()` is skipped resulting in a significant speed-up for larger problems.

In the ALADIN main loop `iterateAL()`, the function `parallelStep()` solves the local NLPs and evaluates the Hessian of the Lagrangian (or it's approximation e.g. when BFGS is used), the gradient of the objective, and the Jacobian of the active constraints (sensitivities) at the NLP's solution. The set of active constraints is determined by primal active set detection described in Appendix A.1. Furthermore, regularization is also

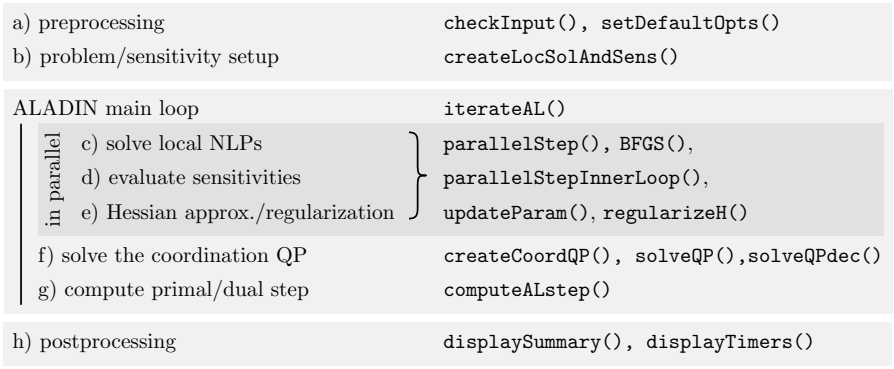



Figure 3: Structure of `run_ALADIN()` in ALADIN-$\alpha$.

done in `parallelStep()` if needed. Moreover, in case the nullspace method or bi-level ALADIN is used, the computation of the nullspcaces and the Schur-complements is also done locally shifting substantial computational burden from the centralized coordination step to `parallelStep()`. The function `updateParam()` computes dynamically changing ALADIN parameters for numerical stability and speedup.

The coordination QP is constructed in the function `createCoordQP()`. Different QP formulations are possible: here we use a variant considering slack variables from Houska et al. [31] for numerical stability. Different dense and sparse solvers for solving the coordination QP are available in `solveQP()`. Most of them are based on solving the first-order necessary conditions which is a linear system of equations. Available solvers are the MATLAB linear-algebra routines `linsolve(), pinv()`, the backslash operator and `MA57` based on the MATLAB LDL-decomposition routine. Using sparse solvers can speed up the computation time substantially. Note that only `MA57` and the backslash-operator support sparse matrices. The solver can be specified by setting the `solveQP` option. In case of convergence problems from remote starting points, it can help to reduce the primal-dual stepsize of the QP step by setting the `stepSize` in the options to a values smaller than 1. More advanced step-size selection rules are subject to ongoing and future work.

## 3.2. Data Structures

The main data structure for defining problems in form of (1) is a struct called `sProb`. In this data structure, the objective functions $\{f_i\}_{i\in\mathcal{S}}$ and constraint functions $\{g_i\}_{i\in\mathcal{S}}$ and $\{h_i\}_{i\in\mathcal{S}}$ are collected in cells which are contained in a nested struct called `locFuns`. Furthermore, `sProb` collects lower/upper bounds (1d) in cells called `llbx` and `uubx`. The coupling matrices $\{A_i\}_{i\in\mathcal{S}}$ are collected in `AA`. Optionally, one can also provide NLP solvers and sensitivities—in this case the problem setup in `createLocSolAndSens()` is skipped leading to a substantial speedup in runtime for larger problems. Furthermore, in this way problem setups can be saved to the hard disk and reloaded later. For a minimal working example of ALADIN-$\alpha$, one only needs to specify `ffi` and `AA`. Optionally one

specify initial guesses in `zz0` and initial Lagrange multipliers `lam0`. The second ingredient for ALADIN-$\alpha$ is an `opts` struct. There, one can specify the variant of ALADIN-$\alpha$ and algorithmic parameters. A full list of options with descriptions can be found under [5].

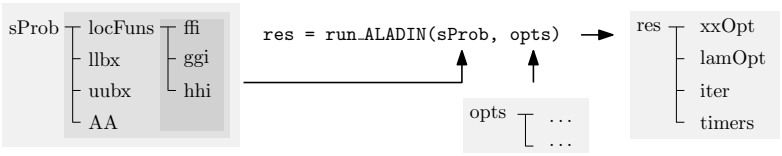



Figure 4: The `sProb` data structure for defining problems in form of (1).

ALADIN-$\alpha$ returns a struct as output. In this struct, one can find a cell of locally primal optimal solutions `xxOpt` containing $\{x_i^\star\}_{i\in\mathcal{S}}$. `lamOpt` contains the optimal lagrange multipliers for the consensus constraints (1e), $\lambda^\star$. Moreover the field `iter` contains information about the ALADIN iterates such as primal/dual iterates and `timers` contains timing information. Note that `run_ALADIN()` and `run_ADMM()` have the same function signature in terms of `sProb`—only the options differ.

## 3.3. Further Features

Next, we describe further features of ALADIN-$\alpha$. These features can be activated/decativated by setting corresponding options. A description of these options can be found under[6].

**Hessian Approximations** Instead of exact Hessians, approximations formulas such as the Broyden-FletcherGoldfarb-Shanno-(BFGS) update can be used either to reduce communication and computational complexity. The BFGS Hessian is activated by setting the `Hess` option either to `BFGS` for standard BFGS or to `DBFGS` for damped BFGS. For details we refer to the book of Nocedal and Wright [33].

**Parametric NLP Setup** A parametric problem setup is possible in the objective functions $f_i$ and the equality/inequality constraints $g_i/h_i$ via the parameters $p_i$. This feature is especially useful in combination with the `reuse` option which returns the internally constructed `CasADi` solvers and derivatives. If one uses this returned problem formulation in a new `run_ALADIN()` run, the problem setup will be skipped which can lead to a substantial speedup. In an MPC setting, for example, the parameter $p_i$ models the changing initial condition during the MPC loops. Moreover, parametric data problem data might also be useful for large-scale problems where one would like to solve an optimization problem for a wide variety of parameters. This feature is accessed by simply adding a parameter cell `p` to the problem structure `sProb` and defining the objective/-constraints in terms of two inputs, $x_i$ and $p_i$. A concrete usage example, also covering the `reuse` option, for distributed predictive control of two mobile robots can be found in [7].

[5] `https://alexe15.github.io/ALADIN.m/options/`
[6] `https://alexe15.github.io/ALADIN.m/options/`
[7] `https://alexe15.github.io/ALADIN.m/robotEx/`

| example | field | `examples/...` | docs alexe15.github.io/ALADIN.m |
|---|---|---|---|
| chemical reactors | chem. eng./control | `chemical_reactor` | |
| mobile robots | robotics/control | `robots` | /robotEx |
| optimal power flow | power systems | `optimal_power_flow` | /redComm |
| sensor network | estimation | `sensor_network` | /ParallelExample |

Table 1: Application examples coming with ALADIN-$\alpha$.

**Parallelization** ALADIN-$\alpha$ also supports parallel computing in case the MATLAB parallel computing toolbox is installed. Here, we exploit the fact that the local NLPs are independent from each other, i.e. they can be solved in parallel. An example for distributed nonlinear estimation with mobile sensor networks can be found under [8]. For this example we get a speedup-factor of about two when using four processors in parallel. Parallel computing can be activated by setting the `parfor` option to `true`.

**Application Examples** In the example collection of ALADIN-$\alpha$ we provide additional numerical examples highlighting applicability of ALADIN-$\alpha$ to a wide range of problems. These example are

- the presented distributed optimal control for a chemical reactor;
- an example for distributed optimal control of mobile robots [1], [35];
- an optimal power flow problem [32] which is one of the most important optimization problems in power systems;
- an example for distributed estimation in mobile sensor networks from Houska et al. [31].

Furthermore, we included several test problems form the Hock-Schittkowski test collection [36]. The code for all these examples is available in the `examples\` folder of ALADIN-$\alpha$. Furthermore, we provide textual descriptions of most of these examples in the documentation of ALADIN-$\alpha$ online under `https://alexe15.github.io/ALADIN.m/`. The examples with their path in ALADIN-$\alpha$ and the corresponding links are summarized in Table 1.

## 4. Illustrative Examples

In this section we provide two numerical examples illustrating how to use ALADIN-$\alpha$ in practice: one minimalistic example showing how to formulate problems in form of (1) and solving this problem with ALADIN-$\alpha$. The second example is an optimal control problem for a three-vessel chemical process where we would highlight the applicability of ALADIN-$\alpha$ to real-life problems. We briefly compare the ALADIN result to ALADIN-$\alpha$'s ADMM implementation.

[8] `https://alexe15.github.io/ALADIN.m/ParallelExample/`

```
% define symbolic variables
y1  =   sym('y1',[1,1],'real');
y2  =   sym('y2',[2,1],'real');

% define symbolic objectives
f1s =   2*(y1-1)^2;
f2s =   (y2(2)-2)^2;

% define symbolic ineq. constraints

h2s = [  -1-y2(1)*y2(2); ...
         -1.5+y2(1)*y2(2)];

% convert symbolic variables to
   MATLAB fuctions
f1 = matlabFunction(f1s,'Vars',{y1});
f2 = matlabFunction(f2s,'Vars',{y2});

h1 = @(y1)[];
h2 = matlabFunction(h2s,'Vars',{y2});
```

```
% define symbolic variables
y_1 = SX.sym('y_1', 1);
y_2 = SX.sym('y_2', 2);

% define symbolic objectives
f1s = 2 * (y_1 - 1)^2;
f2s = (y_2(2) - 2)^2;

% define symbolic ineq. constraints
h1s = [];
h2s = [  -1 - y_2(1)*y_2(2); ...
         -1.5 + y_2(1)*y_2(2)];

% convert symbolic variables to
   MATLAB fuctions
f1 = Function('f1', {y_1}, {f1s});
f2 = Function('f2', {y_2}, {f2s});

h1 = Function('h1', {y_1}, {h1s});
h2 = Function('h2', {y_2}, {h2s});
```

```
% define objectives
f1 = @(y1) 2 * (y1 - 1)^2;
f2 = @(y2) (y2(2) - 2)^2;

% define inequality constraints
h1 = @(y1) []
h2 = @(y2) [-1 - y2(1) * y2(2);...
            -1.5 + y2(1) * y2(2)];
```

Figure 5: Tutorial example with different tools for problem setup.

```
% define coupling matrices
A1 = 1;
A2 = [-1, 0];

% collect problem data in sProb struct
sProb.locFuns.ffi = {f1, f2};
sProb.locFuns.hhi = {h1, h2};

% handing over of coupling matrices to problem
sProb.AA = {A1, A2};

% start solver with default options
sol = run_ALADINnew(sProb);
```

```
============================================================
==              This is ALADIN-alpha v0.1                 ==
============================================================
QP solver:        MA57
Local solver:     ipopt
Inner algorithm:  none

No termination criterion was specified.
Consensus violation: 6.6531e-12

Maximum number of iterations reached.

----------------   ALADIN-alpha timing   ----------------
t[s]                 %tot       %iter
Tot time......:     3.92
Prob setup....:     0.19                4.8
Iter time.....:     3.72                95
---------
NLP time......:     1.1                              29.7
QP time.......:     0.11                              2.8
Reg time......:     0.02                              0.6
Plot time.....:     2.27                             60.8

============================================================
```

Figure 6: Collection of variables (left) and output of ALADIN-$\alpha$ (right).

### 4.1. A Tutorial Example

First, we investigate how to reformulate a tutorial optimization problem in partially separable form 1. Let us consider the non-convex NLP

$$\min_{x_1,x_2\in\mathbb{R}} f(x) = 2\,(x_1-1)^2 + (x_2-2)^2 \tag{2a}$$

$$\text{subject to} \quad -1 \leq x_1 \cdot x_2 \leq 1.5. \tag{2b}$$

In order to apply ALADIN-$\alpha$, problem (2) needs to be in form of (1). To get there, let us introduce auxiliary variables $y_1, y_2$ with $y_1 \in \mathbb{R}$ and $y_2 = (y_{21} \;\; y_{22})^\top$. Let us couple these variables again by introducing a consensus constraint $\sum_i A_i y_i = 0$ with $A_1 = 1$ and $A_2 = (-1 \;\; 0)$. Furthermore, let us reformulate the objective function $f$ by local objective functions $f_1(y_1) := 2\,(y_1-1)^2$ and $f_2(y_2) = (y_{22}-2)^2$ with $f = f_1 + f_2$. Moreover, we reformulate the global inequality constraint (2b) by a local two dimensional constraints function $h_2 = (h_{21} \;\; h_{22})^\top$ with $h_{21}(y_2) = -1 - y_{21}\,y_{22}$ and $h_{22}(y_2) = -1.5 + y_{21}\,y_{22}$. Combining these reformulations yields

$$\min_{y_1\in\mathbb{R},y_2\in\mathbb{R}^2} 2\,(y_1-1)^2 + (y_{22}-2)^2 \tag{3a}$$

$$\text{subject to} \quad -1 - y_{21}\,y_{22} \leq 0, \quad -1.5 + y_{21}y_{22} \leq 0, \tag{3b}$$

$$y_1 \;+\; (-1 \;\; 0)\,y_2 = 0, \tag{3c}$$

which is in form of problem (1). Note that the solutions to (2) and (3) coincide but (3) is of higher dimension, thus one can view the reformulation as a kind of lifting to a space of higher dimensionality. Moreover, observe that this reformulation contains a general strategy for reformulating problems in form of (1): if there is nonlinear coupling in the objective functions or the constraints, introducing auxiliary variables and enforcing them to coincide by an additional consensus constraint in form of (1e) yields purely affine coupling. With that strategy, one can reformulate most nonlinear program in form of (1e).

**Solution with ALADIN-$\alpha$** To solve (3) with ALADIN-$\alpha$, we set up our problem formulation in a struct `sProb` as described in Subsection 3.2. To illustrate different possibilities of problem setup for ALADIN-$\alpha$, we construct the objective and constraints functions in three different ways: a), via the MATLAB symbolic toolbox, b), via the `CasADi` symbolic framework and, c), directly via function handles. All these ways are shown in Figure 5.

After defining objective and constraint functions, all function handles and the coupling matrices $A_i$ are collected in the struct `sProb` (Figure 6). We call the `run_ALADIN()` function with an empty options struct leading to computation with default parameters. These steps and the resulting ALADIN-$\alpha$ report after running `run_ALADIN()` is shown on the right pane of Figure 6. In the ALADIN-$\alpha$ report, the reason for termination and timing of the main ALADIN-$\alpha$ steps is displayed. Note that plotting takes a substantial amount of time—so it advisable to deactivate online plotting if it is not needed for

diagnostic reasons. Figure 7 shows the plotted figures while ALADIN-$\alpha$ is running. The figures show (in this order) the consensus violation $\|Ax - b\|_\infty$, the local step sizes $\|x^k - z^k\|_\infty$, the step size in the coordination step $\|\Delta x^k\|_\infty$ and the changes in the active set. From these figures one usually can recognize divergence quite fast and also can get a feeling on the effectiveness e.g. for new internal heuristics or the degree of accuracy reached after a certain number of iterations.

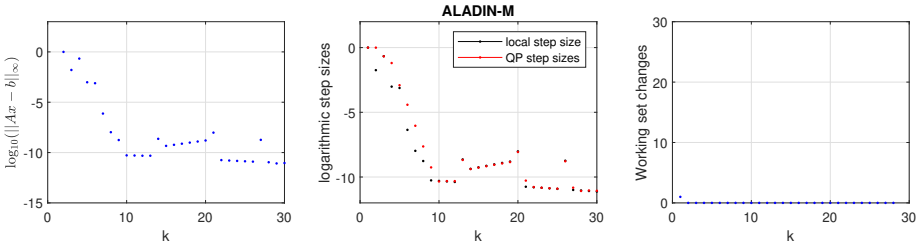



Figure 7: ALADIN-$\alpha$ iteration plot for tutorial problem (3).

## 4.2. Distributed Optimal Control of two Chemical Reactors

Now let us consider a discrete-time optimal control problem (OCP) for a chemical reactor-separator process. This OCP can serve as a basis for distributed model predictive control [37]–[39]. The chemical process we consider here consists of two CSTRs and a flash separator as shown in Figure 8 [40], [41]. The goal is to steer the reactor to the optimal setpoint from an initial point $x(0)^\top = (360.69 \;\; 3.19 \;\; 0.15 \;\; 0.03 \;\; 430.91 \;\; 2.76 \;\; 0.34 \;\; 0.08 \;\; 430.42 \;\; 2.79 \;\; 0.38 \;\; 0.08)$. After applying a fourth-order Runge-Kutta scheme for ODE discretization, the dynamics of the CSTRs and the flash separator are given by

$$x_i^{k+1} = q_i(x_i^k, u_i^k, z_i^k) \quad \text{for} \quad i \in \mathcal{S},$$

where $q_i : \mathbb{R}^{n_{xi}} \times \mathbb{R}^{n_{ui}} \times \mathbb{R}^{n_{zi}} \to \mathbb{R}^{n_{xi}}$ are the dynamics of the $i$th vessel and where $\mathcal{S} := \{1, 2, 3\}$ is the set of vessels. Here, $x_i^\top = (x_{Ai}, x_{Bi}, x_{Ci}, T_i)$ are the states with $x_{Ai}, x_{Bi}$ ($10^3$ mol/m$^3$) being the concentrations of the reactants, $A$, $B$ and $C$ and $T$ (K) is the temperature. The inputs $u_i = Q_i$ ($10^3$ J/h) denote the heat-influxes of the individual vessel and $z_i$ are copied states of the neighbored reactors influencing reactor $i$, i.e. $z_i := (x_j)_{j \in N(i)}$. Note that the feed-stream flow rates $F_{10}, F_{20}, F_3, F_R$ and $F_p$ are fixed and given. Detailed equations for the dynamics of the CSTRs/separator are given in Christofides et al. [41]. With the above, we are ready to formulate a discrete-time

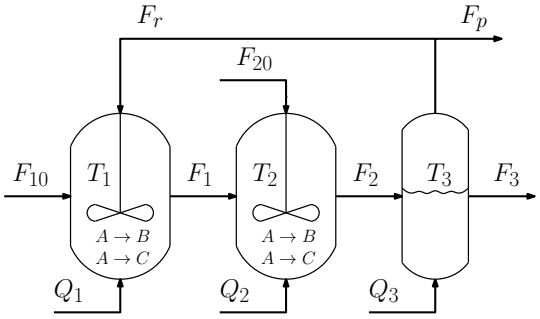



Figure 8: Reactor-separator process.

optimal control problem

$$\min_{\substack{(x_i^k,\, z_i^k, u_i^k) \\ \forall k \in \mathbb{I}_{[1\ T]} \\ \forall i \in \mathcal{S}}} \sum_{i \in \mathcal{S}} \sum_{k \in \mathbb{I}_{[1\ T]}} \frac{1}{2}(x_i^k - x_{is})^\top Q_i\,(x_i^k - x_{is}) + \sum_{k \in \mathbb{I}_{[1\ T-1]}} \frac{1}{2}(u_i^k - u_{is})^\top R_i\,(u_i^k - u_{is}^k) \quad \text{(4a)}$$

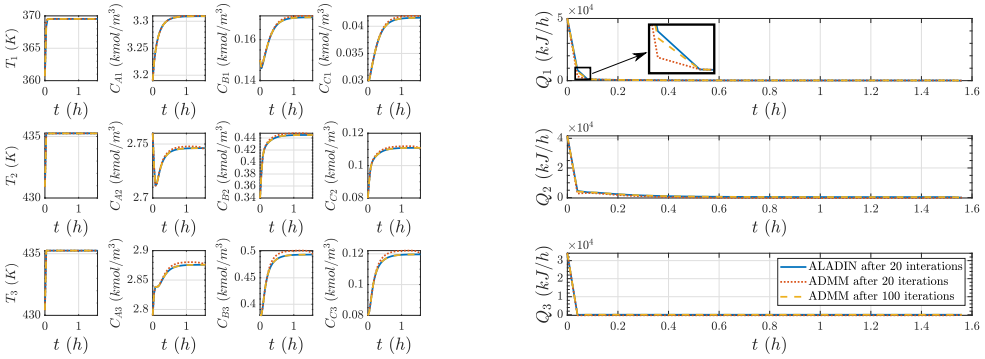
$$\text{s.t.} \quad x_i^{k+1} - q_i(x_i^k, u_i^k, z_i^k) = 0, \quad x_i^0 = x_i(0) \quad \text{for all } k \in \mathbb{I}_{[1\ T]} \text{ and for all } i \in \mathcal{S}, \quad \text{(4b)}$$

$$\underline{u}_i \le u_i^k \le \bar{u}_i \quad \underline{x}_i \le x_i^k \quad \text{for all } k \in \mathbb{I}_{[1\ T]} \text{ and for all } i \in \mathcal{S}, \quad \text{(4c)}$$

$$\sum_{i \in \mathcal{S}} A_i \begin{pmatrix} x_i^{k\top} & z_i^{k\top} & u_i^{k\top} \end{pmatrix}^\top = 0 \quad \text{for all } k \in \mathbb{I}_{[1\ T]}, \quad \text{(4d)}$$

with lower/upper bounds on the inputs $\overline{u} = -\underline{u} = (5 \cdot 10^4 \quad 1.5 \cdot 10^5 \quad 2 \cdot 10^5)^\top$ and lower bounds on the states $\underline{x}_i^k = 0$ for all times $k \in \mathbb{I}_{[1\ T]}$ and all vessels $i \in \mathcal{S}$. The weighting matrices are chose to $Q_i = \mathrm{diag}(20 \quad 10^3 \quad 10^3 \quad 10^3)$ and $R_i = 10^{-10}$. The matrices $A_i$ are selected such that they represent the constraint $z_i := (x_j)_{j \in N(i)}$. The sampling time is $\Delta h = 0.01h$ and the horizon is $T = 10\,\mathrm{h}$. By defining $\tilde{x}_i^\top := \begin{pmatrix} x_i^{k\top} & z_i^{k\top} & u_i^{k\top} \end{pmatrix}_{k \in \mathbb{I}_{[1\ T]}}$, $f_i(\tilde{x}_i) := \sum_{k \in \mathbb{I}_{[1\ T]}} \frac{1}{2}(x_i^k - x_{is})^\top Q_i\,(x_i^k - x_{is}) + \sum_{k \in \mathbb{I}_{[1\ T-1]}} \frac{1}{2}(u_i^k - u_{is})^\top R_i\,(u_i^k - u_{is}^k)$, $g_i(\tilde{x}_i) := \left(x_i^{k+1} - q_i(x_i^k, u_i^k, z_i^k)\right)_{k \in \mathbb{I}_{[1\ T-1]}}$, and $h_i(\tilde{x}_i) := \left((\underline{u}_i - u_i^k \quad u_i^k - \bar{u}_i \quad \underline{x}_i - x_i^k)^\top\right)_{k \in \mathbb{I}_{[1\ T]}}$ one can see that the OCP (4) is in form of (1) and thus solvable by ALADIN-$\alpha$ ($\tilde{x}_i$ here corresponds to $x_i$ in (1)).

**Numerical Results** Figure 9 shows the resulting input and state trajectories for one OCP (4) for ALADIN and ADMM after 20 iterations, and for ADMM after 100 iterations. At first-glance all trajectories are quite close to each other. However, small differences in the input trajectories can be observed. When having a closer look at Figure 10 showing convergence measure over the iteration index $k$, one can see that in logarithmic scale the differences can be quite large. Fore example the consensus gap $\|Ax - b\|_\infty$ is in an order of $10^1$ after 20 iterations which means that the physical values at the interconnection points have a maximum mismatch of $10^1$. ALADIN converges quite fast and also to a very high accuracy but needs more communication in each step.




(a) States (b) Inputs

Figure 9: Optimal state/input trajectories computed by ALADIN & ADMM.

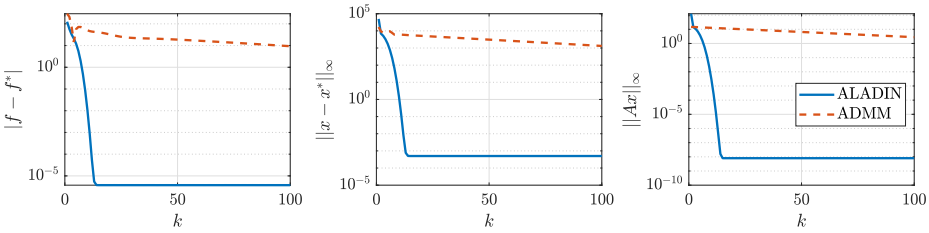



Figure 10: Convergence of ALADIN (solid) and ADMM (dashed) for problem (4).

All trajectories were computed with `run_ALADIN()` and `run_ADMM()`. The numerical implementation is part of the example collection of ALADIN-$\alpha$; all parameters and the reactor dynamics are given there.

## 5. Conclusion & Future Work

We presented one of the first open source toolboxes for distributed non-convex optimization. Our toolbox ALADIN-$\alpha$ is based on the Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN) algorithm and implements various extensions to standard ALADIN mostly aiming at reducing communication and coordination overhead in ALADIN. Moreover, ALADIN-$\alpha$ comes with a rich set of application examples from different engineering fields reaching from power systems over non-linear control to mobile sensor networks.

Although ALADIN-$\alpha$ performs very well for many small to medium-sized problems, we aim at further improving numerical stability in future work by developing more advanced internal auto-tuning routines. Furthermore, developing distributed globalization routines for enlarging the set of possible initializations seems important and promising. Code generation for distributed optimization on embedded devices is another possible

research direction.

**Acknowledgement** We would like to thank Tillmann Mühlpfordt for very helpful discussions and suggestions, and Veit Hagenmeyer for strongly supporting the development of ALADIN-$\alpha$. Moreover, Timm Faulwasser acknowledges financial support by the Elite Program for Postdocs of the Baden-Wrttemberg Stiftung.

# A. Implementation Details

In this appendix we describe additional details on the implementation of ALADIN-$\alpha$. The main algorithm is based on Houska et al.[31], but additional practical considerations have been taking into account improving efficiency and numerical stability of ALADIN.

## A.1. ALADIN in Detail

The here-used version of standard ALADIN is summarized in Algorithm 1. Each ALADIN iteration executes three main steps: Step 1 solves local NLPs (5) for fixed and given values for primal iterates $z_i^k$ and dual iterates $\lambda^k$ in parallel. The parameter sequences $\{\Sigma_i^k\} \succ 0$ are user-defined—details are described inSubsection A.4.[9] Note that the equality constraints $g_i$ and box constraints are not explicitly detailed in Houska et al.[31]—we consider them separately here for numerical efficiency reasons.[10] Step 2 computes local sensitivities such as the gradient of the local objectives $\nabla f_i(x_i^k)$ and positive definite approximations of the local Hessian matrices $B_i^k \approx H_i^k = \nabla_{xx}^2\{f_i(x_i^k) + \kappa_i^{k\top} g_i(x_i) + \left(\gamma_i^{k\top} \;\; \eta_i^{k\top}\right)_{j\in\mathcal{A}_i^k}\left(\tilde{h}_j(x_i^k)\right)_{j\in\mathcal{A}_i^k}\}$. Here, we define combined inequality constraints $\tilde{h}(x)^\top := \left(h(x)^\top \;\; (\underline{x}_i - x_i)^\top \;\; (x_i - \bar{x}_i)^\top\right)$ and a *set of active inequality constraints* in subproblems $i \in \mathcal{S}$

$$\mathbb{A}_i^k := \left\{ j \in \{1, \ldots, n_{hi} + 2n_{xi}\} \mid \left(\tilde{h}(x^k)\right)_j > -\tau \right\},$$

with $\tau > 0$ being a user-defined parameter which can be specified via the `actMargin` option. Moreover, the Jacobians of the active constraints are defined as $C_i^{k\top} := \left(\nabla g_i(x_i^k)^\top \;\; \left(\nabla \tilde{h}_j(x_i^k)\right)_{j\in\mathbb{A}_i^k}^\top\right)$. With this sensitivity information, step 4 solves an equality constrained quadratic program (6) serving as a coordination problem. The last step 5 of Algorithm 1 updates $z^k$ and $\lambda^k$ based on the solution to (6). In order to get global convergence, the step size parameter $\alpha \in (0 \;\; 1]$ has to be properly chosen by a globalization routine. Designing suitable distributed globalization routines is subject of ongoing and future work, currently we use the full step variant with $\alpha = 1$. A smaller stepsize can be specified via the `stepSize` option which might stabilize ALADIN-$\alpha$ for certain problems. Note that time-varying parameter sequences $\{\Delta^k\}$ and $\{\Sigma_i^k\}$ with $\Sigma_i^k, \Delta^k \succ 0$

[9] We use scaled 2-norms $\|x\|_\Sigma := \sqrt{x^\top \Sigma x}$ here.

[10] Some numerical solvers for (5) can for example treat box constraints in an efficient way by using projection methods.

**Algorithm 1** Augmented Lagrangian Alternating Direction Inexact Newton (ALADIN)

**Initialization:** Initial guess $\left(\{z_i^0\}_{i\in\mathcal{S}}, \lambda^0\right)$, choose $\Sigma_i \succ 0, \rho^0, \mu^0, \epsilon$.
**Repeat:**

1. *Parallelizable Step:* For each $i \in \mathcal{S}$, solve

$$\min_{x_i} f_i(x_i) + (\lambda^k)^\top A_i x_i + \left\|x_i - z_i^k\right\|_{\Sigma_i^k}^2 \tag{5a}$$

$$\text{subject to } g_i(x_i) = 0,\ h_i(x_i) \leq 0, \text{ and } \underline{x}_i \leq x_i \leq \overline{x}_i. \tag{5b}$$

2. *Termination Check:* If $\left\|\sum_{i\in\mathcal{S}} A_i x_i^k - b\right\| \leq \epsilon$ and $\left\|x^k - z^k\right\| \leq \epsilon$ , return $x^\star = x^k$.

3. *Sensitivity Evaluations:* Compute and communicate local gradients $g_i^k = \nabla f_i(x_i^k)$, Hessian approximations $0 \prec B_i^k \approx \nabla^2\{f_i(x_i^k) + \kappa_i^{k\top} h_i(x_i^k)\}$ and constraint Jacobians $C_i^{k\top} := \left[\nabla g_i(x_i^k)^\top \ \left(\nabla \tilde{h}_i(x_i^k)\right)_{j\in\mathbb{A}^k}^\top\right]$.

4. *Consensus Step:* Solve the coordination QP

$$\begin{aligned} \min_{\Delta x, s} \ & \sum_{i\in\mathcal{S}} \frac{1}{2}\Delta x_i^\top B_i^k \Delta x_i + g_i^{k\top}\Delta x_i + (\lambda^k)^\top s + \|s\|_{\Delta^k}^2 \\ \text{subject to } & \sum_{i\in\mathcal{S}} A_i(x_i^k + \Delta x_i) = s \qquad \mid \lambda^{\mathrm{QP}k}, \\ & C_i^k \Delta x_i = 0 \qquad \forall i \in \mathcal{S}, \end{aligned} \tag{6}$$

   returning $\Delta x^k$ and $\lambda^{\mathrm{QP}k}$.

5. *Line Search:* Update primal and dual variables by

$$z^{k+1} \ \leftarrow \ z^k + \alpha^k \Delta x^k \qquad\qquad \lambda^{k+1} \leftarrow \lambda^k + \alpha^k(\lambda^{\mathrm{QP}k} - \lambda^k),$$

   with $\alpha^k = 1$ for a full-step variant. Update $\Sigma_i^k$ and $\Delta^k$.

| name | MA57 | pinv | linsolve |
|---|---|---|---|
| algorithm | multifrontal LDL | based on SVD | LU |
| sparse | yes | yes | no |

Table 2: Centralized QP solvers interfaced in ALADIN-$\alpha$.

might accelerate convergence of ALADIN in practice. Heuristic routines for doing so are described in Subsection A.4.[11]

## A.2. Solving the Coordination QP

Note that as $B_i^k$ is assumed to be positive definite, (6) is a strictly convex equality-constrainted QP which can be equivalently solved by solving the first order optimality conditions (if $C_i^k$ has full row rank) which is a linear system of equations. The coordination problem (6) can be solved in two different ways: either by centralized linear algebra routines or by iterative methods. For centralized computation, several solvers are interfaced in ALADIN-$\alpha$ which can be specified via the `solveQP` option. The available solvers are summarized in Table 2. Note that not all solvers support sparse matrices. MA57 usually perfoms very well in practice—both in terms of speed and robustness. The second approach to solve (6) is via iterative and possibly *decentralized* routines. Details of these decentralized routines are described in Subsection A.6.

## A.3. Hessian Approximations

As $H_i^k$ may have zero eigenvalues or may even be indefinite, special care has to be taken to ensure it's positive definiteness.[12] Here we use a heuristic using ideas from Nocedal and Wright [33]—other heuristics are possible and might accelerate convergence. Our heuristics "flips" the sign of the negative eigenvalues (if there are any) and puts the zero eigenvalues to a small positive number $\delta$. The intuition here is that the stepsize in the direction of negative curvature gets smaller the "more negative" the curvature is; we made good numerical experience with that rule. However there is no more sophisticated mathematical reasoning behind. Mathematically this means we compute an eigendecomposition $H_i^k = V_i \Lambda_i V_i^\top$ locally, where $\Lambda_i$ is a matrix with the eigenvalues of $H_i$ on its main diagonal and $V_i$ is a matrix of the corresponding eigenvectors. Our rule

[11] Note that in contrast to Houska et al. [31], we omit the term $\rho/2$ in front of the penalization term in (5) removing redundancy. The setting from Houska et al. [31] can be recovered by choosing $\Sigma_i^k = \rho^k/2\, I$.

[12] If one would use $\nabla^2\{f_i(x_i^k) + \kappa_i^{k\top} h_i(x_i^k)\}$ regardless, the coordination step (6) would not necessarily produce decent directions destroying the local convergence properties of ALADIN. In case of zero eigenvalues, $\nabla^2\{f_i(x_i^k) + \kappa_i^{k\top} h_i(x_i^k)\}$ is singular and the coordination step can not be easily solved by a standard solver for linear systems of equations.

for regularization is

$$\tilde{\Lambda}_{jj} := \begin{cases} |\Lambda_{jj}| & \text{if } \Lambda_{jj} < -\delta, \\ \delta & \text{if } |\Lambda_{jj}| \in (-\delta,\ \delta), \\ \Lambda_{jj} & \text{else,} \end{cases} \qquad \text{and} \qquad B_i^k := V_i \tilde{\Lambda}_i V_i^\top,$$

with $\delta = 10^{-4}$. Regularization and the corresponding $\delta$ can be activated and specified by the options `reg` and `regParam` corresponding to $\delta$.

As an alternative to exact Hessians with regularization, one can use the Broyden-FletcherGoldfarb-Shanno (BFGS) update-rule for successively approximating the exact Hessian based on the gradient of the Lagrangian. This has the advantage that only the gradient of the Lagrangian has to be communicated (which is a vector) instead of the Hessian (which is a matrix). A detailed description on how to use BFGS within ALADIN can be found in Engelmann et al. [32]. The BFGS formula can be activated by the setting the option `Hess` to `BFGS` or to `DBFGS` for damped BFGS. The advantage of damped-BFGS is that it guarantees positive-definiteness of $B_i^k$ regardless of the positive-definiteness of the exact Hessian at the current iterate. Note that in case the nullspace method which we will introduce in Subsection A.5 is used, the regularization is done for the reduced Hessian $\bar{H}_i^k$ instead of $H_i^k$.

## A.4. Scaling Matrices

A simple Heuristic for the sequences $\{\Sigma_i^k\} \succ 0$ and $\{\Delta^k\} \succ 0$ is to start with certain (usually diagonal) initial matrices $\Sigma_i^0, \Delta^0$ and to multiply them by a factor $r_\Delta, r_\Sigma > 1$ in each iteration, i.e.

$$\Sigma_i^{k+1} = \begin{cases} r_\Sigma \Sigma_i^k & \text{if } \|\Sigma_i\|_\infty < \bar{\sigma} \\ \Sigma_i^k & \text{otherwise} \end{cases} \qquad \text{and} \qquad \Delta^{k+1} = \begin{cases} r_\Delta \Delta^k & \text{if } \|\Delta\|_\infty < \bar{\delta} \\ \Delta^k & \text{otherwise.} \end{cases} \tag{7}$$

This routine is successfully used in previous works.[1], [32] An alternative option for choosing the sequence $\{\Delta^k\}$ is based on the consensus violation for each individual row in (1e). The idea here is to increase the corresponding $\Delta_{ii}^k$ to drive the corresponding consensus violation to zero. This technique is common in algorithms based on augmented Lagrangians, cf. [42, Chap 4.2.2]. Mathematically this means that we choose

$$\Delta_{cc}^{k+1} = \begin{cases} \beta\, \Delta_{cc}^k & \text{if } \left|\left(\sum_{i\in\mathcal{S}} A_i x_i^k - b\right)_c\right| > \gamma \left|\left(\sum_{i\in\mathcal{S}} A_i x_i^{k-1} - b\right)_c\right| \\ \Delta_{cc}^k & \text{if } \left|\left(\sum_{i\in\mathcal{S}} A_i x_i^k - b\right)_c\right| \leq \gamma \left|\left(\sum_{i\in\mathcal{S}} A_i x_i^{k-1} - b\right)_c\right| \end{cases} \text{for all } c \in 1, \ldots, n_c,$$

with $\gamma \in (0\ \ 1)$ and $\beta > 1$. In ALADIN-$\alpha$ we choose $\beta = 10$ and $\gamma = 0.25$. This rule can be activated by the option `DelUp` and is able to accelerate convergence of ALADIN-$\alpha$ substantially in some cases.

Note that all the above heuristics such as regularization or parameter updates do—although they are heuristics—not interfere with the fast local convergence properties of

ALADIN-$\alpha$. They are rather required for guaranteeing fast local convergence since they ensure that the assumptions made in the local convergence proof of ALADIN such as the positive-defniteness of $B_i^k$ are satisfied [31].

## A.5. The Nullspace Method

The nullspace method can be used to reduce the dimensionality of the coordination QP (6), thus reducing communication and computation in the coordination step. The idea here is to parameterize the nullspace of the active constraints $\text{null}(C_i^k) := \{x_i \in \mathbb{R}^{n_{xi}} \mid C_i^k x_i^k = 0\}$ by $\text{null}(C_i^k) = Z_i^k \Delta v_i$, where $Z_i^k \in \mathbb{R}^{x_{xi} \times \left(n_{xi} - |\mathbb{A}_i^k|\right)}$ is matrix which's columns form a basis of $\text{null}(C_i^k)$. Note that $C_i^k Z_i^k = 0$ by definition of the nullspace. By using this parametrization, (6) can be written as

$$\begin{aligned} \min_{\Delta v, s} \ & \sum_{i \in \mathcal{S}} \left\{ \frac{1}{2} \Delta v_i^\top \bar{B}_i^k \Delta v_i + \bar{g}_i^{k\top} \Delta v_i \right\} + (\lambda^k)^\top s + \|s\|_{\Delta^k}^2 \\ \text{subject to } & \sum_{i \in \mathcal{S}} \bar{A}_i^k (v_i^k + \Delta v_i) = s \qquad \mid \lambda^{\text{QP}k}, \end{aligned} \tag{8}$$

where $\bar{H}_i^k = Z_i^{k\top} H_i^k Z_i^k \in \mathbb{R}^{\left(n_{xi} - |\mathbb{A}_i^k|\right) \times \left(n_{xi} - |\mathbb{A}_i^k|\right)}$, $\bar{g}_i^k = Z_i^{k\top} g_i^k \in \mathbb{R}^{\left(n_{xi} - |\mathbb{A}_i^k|\right)}$ and $\bar{A}_i^k = A_i Z_i^k \in \mathbb{R}^{n_c \times \left(n_{xi} - |\mathbb{A}_i^k|\right)}$. Note that $\bar{A}_i^k$ has an iteration index $k$ and changes during the iterations since $Z_i^k$ changes. Similar to the full-space approach, regulaization from Subsection A.3 is also used here (if it is activated via the option reg `reg`) yielding a positive definite $\bar{B}_i^k$. The nullspace method can be used by activating the option `redSpace`. Notice that (at least in case of dense matrix storage formats) the required communication between the subproblems and the coordinator is reduced by twice the number of equality constraints and active inequality constraints. The amount of communication reduction can be quite large—especially in case of problems with a large number of constraints. Furthermore, the coordination QP (6) is in general less expensive to solve since (8) is of smaller dimension than (6). Moreover, (8) is strongly convex under suitable assumptions which (6) is not necessarily [1]. Note that computing such a nullspace is expensive in general as it involves a singular-value decomposition. However, the advantage here is that this step is done *locally*, enabling parallel computation.

## A.6. Bi-level ALADIN

Bi-level ALADIN is and extension of ALADIN to further reduce dimensionality of the already reduced coordination QP (8) and thereby further reducing communication and computation overhead of the coordination step. Moreover, bi-level ALADIN uses *decentralized* inner algorithms for solving further reduced coordination QP. By doing so, ALADIN becomes a *decentralized* optimization algorithm which means that it can operate purely on a neighbor-to-neighbor basis. Note that the here-presented inner algorithms *decentralized ADMM* (D-ADMM) and *decentralized conjugate gradients* (D-CG) are improved versions of the algorithms given in Engelmann et al. [1]. Their derivation is more intuitive and, in case of D-CG, the version given here *avoids* the preparation step from

Engelmann et al. [1] further improving communication requirements. Nonetheless—in terms of their theoretical properties and the generated iteration sequences—the algorithms from Engelmann et al. [1] and the here-given variants are equivalent. We provide a detailed derivation of both of these two algorithms in Engelmann et al. [1].

Let us briefly introduce the main idea of bi-level ALADIN. Writing down the KKT conditions for (8) yields (under the assumptions from Engelmann et al.[1])

$$\bar{B}^k \Delta v + \bar{g}^k + \bar{A}^{k\top} \lambda^{\mathrm{QP}} = 0, \tag{9a}$$

$$\bar{A}^k (v^k + \Delta v) - b - \frac{1}{\mu}(\lambda^{\mathrm{QP}} - \lambda^k) = 0, \tag{9b}$$

where $\bar{B}^k$, $\bar{A}^k$ and $\bar{\Delta} v^k$ are block-diagonal concatenations of $\bar{B}_i^k$, $\bar{A}_i^k$ and $\bar{\Delta} v_i^k$. Using the *Schur-complement* reveals that (9) is equivalent to solving the unconstrained and strongly convex QP

$$\left( \sum_{i \in \mathcal{S}} S_i^k + \frac{1}{\mu} I \right) \lambda^{\mathrm{QP}} = \sum_{i \in \mathcal{S}} s_i^k + I \left( \frac{1}{\mu} \lambda^k - b \right), \tag{10}$$

where we have local Schur-complement matrices $S_i^k := \bar{A}^k \, \bar{B}^{k^{-1}} \bar{A}^{k\top}$ and vectors $s_i^k := \bar{A}^k \left( v^k - \bar{B}^{k^{-1}} \bar{g}^k \right)$, cf. Engelmann et al. [1]. The key observation for decentralization is that the matrices $S_i^k$ and vectors $s_i^k$ inherit the sparsity pattern of the consensus matrices $A_i$, i.e. zero rows in $A_i$ yield zero rows/columns in $S_i^k$ and $s_i^k$. Intuitively speaking, each row/column of $S_i^k$ corresponds to one consensus constraint (row of (1e)) and only the subproblems which "participate" in this constraint have non-zero rows in their corresponding $S_i$. Mathematically, we collect all indices for which $S_i$ has non-zero rows/columns in the set $\mathcal{C}(i)$. Note this set corresponds to all non-zero rows of $A_i$, which can be directly shown from the definition of $S_i^k$ [1]. That means that we can equivalently define $\mathcal{C}(i) := \{c \in \{1, \dots, n_c\} \mid A_{ic} = 0\}$, where $A_{ic}$ denotes the $c$th row of $A_i$. We say that two subproblems $i, j \in \mathcal{S}$ are *neighbored*, if $\mathcal{C}(i) \cap \mathcal{C}(j) \neq \emptyset$.

In order to exploit this sparsity, we introduce matrices $I_{\mathcal{B}} := \left( e_s^\top \right)_{s \in \mathcal{B}} \in \mathbb{R}^{|\mathcal{B}| \times n_c}$, where $e_c \in \mathbb{R}^{n_c}$ is the $c$th unit-vector and $\mathcal{B}$ is an arbitrary index set. Moreover, let us define $I_{ij} = I_{\mathcal{C}(i)} I_{\mathcal{C}(j)}^\top$, $\Lambda_i = I_{\mathcal{C}(i)} \left( \sum_{j \in \mathcal{S}} I_{\mathcal{C}(j)}^\top I_{\mathcal{C}(j)} \right) I_{\mathcal{C}(i)}^\top$, $\hat{S}_i^k = I_{\mathcal{C}(i)}^\top \tilde{S}_i^k I_{\mathcal{C}(i)}$, and $\hat{s}_i^k = I_{\mathcal{C}(i)} \tilde{s}^k$. Based on these definitions, it is possible to derive D-CG and D-ADMM. The resulting algorithms are summarized in Algorithm 2 (D-ADMM) and Algorithm 3 (D-CG). Therein, superscripts $(\cdot)^n$ denote inner iterations. Detailed derivations can be found in Engelmann et al. [1], [43]. In both algorithms, we have three different types of steps: *Local* steps which are independently performed by each subsystem $i \in \mathcal{S}$, *neighbor-neighbor* steps requiring communication between neighbors,[13] and *scalar global sums* (only in D-CG), which requires summing up one scalar number in each inner iteration globally. D-ADMM maintains the variables $\lambda_i, \bar{\lambda}_i \in \mathbb{R}^{|\mathcal{C}(i)|}$, and $\gamma_i \in \mathbb{R}^{|\mathcal{C}(i)|}$ locally in each subsystem $i \in \mathcal{S}$. D-CG has $\lambda_i, r_i, u_i \in \mathbb{R}^{|\mathcal{C}(i)|}$ and $p_i \in \mathbb{R}^{|\mathcal{C}(i)|}$ as local variables.

[13] It can be shown that if two subproblems $i, j \in \mathcal{S}$ are *not* neighbored, we have $I_{ij} = 0$ making D-CG and D-ADMM decentralized.[43]

**Algorithm 2** D-ADMM for problem (10)

**Initialization:** $\bar{\lambda}_i^0$, $\gamma_i^0$ **for all** $i \in \mathcal{S}$, $\rho$
**Repeat for all** $i \in \mathcal{S}$**:**

1. $\lambda_i^{n+1} = \left(\hat{S}_i^k + \rho I\right)^{-1} \left(\hat{s}_i^k - \gamma_i^n + \rho \bar{\lambda}_i^n\right)$ (local)
2. $\bar{\lambda}_i^{n+1} = (\Lambda_i)^{-1} \sum_{j \in N(i)} I_{ij} \lambda_j^{n+1}$ (neighbor-neighbor)
3. $\gamma_i^{n+1} = \gamma_i^n + \rho(\lambda_i^{n+1} - I_{\mathcal{C}(i)} \bar{\lambda}^{n+1})$ (local)

**Algorithm 3** D-CG for problem (10)

**Initialization:** $\lambda^0$ **and** $r^0 = p^0 = \tilde{s} - \tilde{S}\lambda^0$
**Repeat for all** $i \in \mathcal{S}$**:**

1. $\sigma^n = \sum_{i \in \mathcal{S}} \sigma_i^n$ (scalar global sum)
2. $r_i^{n+1} = r_i^n - \frac{\eta^n}{\sigma^n} \sum_{i \in N(i)} I_{ji} u_j$ (neighbor-neighbor)
3. $\eta_i^{n+1} = r_i^n \Lambda_i r_i^n \quad \lambda_i^{n+1} = \lambda_i^n + \frac{\eta^n}{\sigma^n} p_i$ (local)
4. $\eta^{n+1} = \sum_{i \in \mathcal{S}} \eta_i^{n+1}$ (scalar global sum)
5. $p_i^{n+1} = r_i^n + \frac{\eta^{n+1}}{\eta^n} p_i^n \quad u_i^{n+1} = \hat{S}_i p_i^{n+1} \quad \sigma_i^{n+1} = p_i^{n+1\top} \hat{S}_i p_i^{n+1}$ (local)

The amount of information exchanged between neighbors is almost the same in both algorithms. In D-ADMM, the product $I_{ij}\lambda_i$ is exchanged comprising $|\mathcal{C}(i)|$ floats, which is the same number of floats required for the product $I_{ij}u_i$. However, D-CG requires to exchange two scalar values per subsystem in each iteration additionally. On the other hand this yields to substantial acceleration of convergence—theoretically and practically [1]. The information exchange and the locally maintained variables for both algorithms are graphically illustrated in Figure 11. In ALADIN-$\alpha$, both algorithms D-ADMM and D-CG terminate after a fixed number of inner iterations which can be specified via the `innerIter` option. Furthermore, warm starting of both algorithms with the previous $\lambda$ improves the overall performance significantly and is by default activated in the `warmStart` option. Note that D-CG has to be initialized *consistently*, i.e. the initial values $r^0$ and $p^0$ have to fulfill the condition in the initialization step of Algorithm 3. The type of inner algorithm can be specified with the `innerAlg` option; the parameter $\rho$ for ADMM is specified with `rhoADM`.

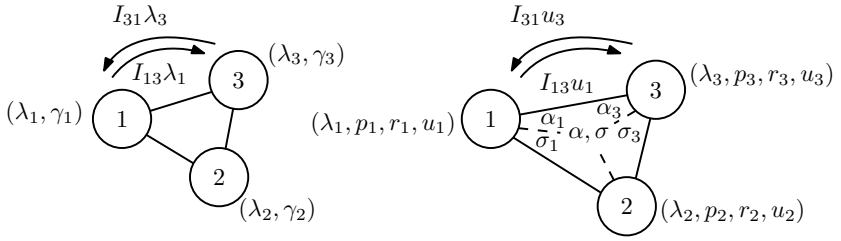



Figure 11: Information exchange and local variables for D-ADMM and D-CG.

## References


[1] A. Engelmann, Y. Jiang, B. Houska, and T. Faulwasser, "Decomposition of non-convex optimization via bi-level distributed ALADIN," *Conditionally accepted for the IEEE Transactions on Control of Network Systems*, 2020.

[2] T. Erseghe, "Distributed Optimal Power Flow Using ADMM," *IEEE Transactions on Power Systems*, vol. 29, no. 5, pp. 2370–2380, 2014.

[3] B. Kim and R. Baldick, "A comparison of distributed optimal power flow algorithms," *IEEE Transactions on Power Systems*, vol. 15, no. 2, pp. 599–604, 2000.

[4] E. Dall'Anese, H. Zhu, and G. B. Giannakis, "Distributed Optimal Power Flow for Smart Microgrids," *IEEE Transactions on Smart Grid*, vol. 4, no. 3, pp. 1464–1475, 2013.

[5] Y. Jiang, M. Zanon, R. Hult, and B. Houska, "Distributed Algorithm for Optimal Vehicle Coordination at Traffic Intersections," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 577–11 582, 2017.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends{\textregistered} in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[7] M. J. Tippett and J. Bao, "Distributed model predictive control based on dissipativity," en, *AIChE Journal*, vol. 59, no. 3, pp. 787–804, 2013.

[8] P. D. Christofides, R. Scattolini, D. Muñoz de la Peña, and J. Liu, "Distributed model predictive control: A tutorial review and future research directions," en, *Computers & Chemical Engineering*, CPC VIII, vol. 51, pp. 21–41, 2013.

[9] B. T. Stewart, A. N. Venkat, J. B. Rawlings, S. J. Wright, and G. Pannocchia, "Cooperative distributed model predictive control," en, *Systems & Control Letters*, vol. 59, no. 8, pp. 460–469, 2010.

[10] Q. Shi, C. He, H. Chen, and L. Jiang, "Distributed Wireless Sensor Network Localization Via Sequential Greedy Optimization Algorithm," *IEEE Transactions on Signal Processing*, vol. 58, no. 6, pp. 3328–3340, 2010.

[11] X. Du, A. Engelmann, Y. Jiang, T. Faulwasser, and B. Houska, "Distributed State Estimation for AC Power Systems using Gauss-Newton ALADIN," in *Accepted for the IEEE 58th Conference on Decision and Control*, 2019.

[12] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, 2004, pp. 20–27.

[13] S. Kar, J. M. F. Moura, and K. Ramanan, "Distributed Parameter Estimation in Sensor Networks: Nonlinear Observation Models and Imperfect Communication," *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 3575–3605, 2012.

[14] J. Guo, G. Hug, and O. K. Tonguz, "A Case for Nonconvex Distributed Optimization in Large-Scale Power Systems," *IEEE Transactions on Power Systems*, vol. 32, no. 5, pp. 3842–3851, 2017.

[15] J. Gondzio and A. Grothey, "Parallel interior-point solver for structured quadratic programs: Application to financial planning problems," en, *Annals of Operations Research*, vol. 152, no. 1, pp. 319–339, 2007.

[16] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," en, *Mathematical Programming Computation*, vol. 7, no. 3, pp. 289–329, 2015.

[17] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," en, *Mathematical Programming Computation*, 2020.

[18] N. Chiang, C. G. Petra, and V. M. Zavala, "Structured nonconvex optimization of large-scale energy systems using PIPS-NLP," in *2014 Power Systems Computation Conference*, 2014, pp. 1–7.

[19] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala, "Scalable stochastic optimization of complex energy systems," in *SC '11: PROCEEDINGS of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–10.

[20] C. G. Petra, N. Chiang, and M. Anitescu, "A Structured Quasi-Newton Algorithm for Optimizing with Incomplete Hessian Information," *SIAM Journal on Optimization*, vol. 29, no. 2, pp. 1048–1075, 2019.

[21] C. G. Petra, "A memory-distributed quasi-Newton solver for nonlinear programming problems with a small number of general constraints," en, *Journal of Parallel and Distributed Computing*, vol. 133, pp. 337–348, 2019.

[22] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, "PARDISO: A high-performance serial and parallel sparse linear solver in semiconductor device simulation," en, *Future Generation Computer Systems*, I. High Performance Numerical Methods and Applications. II. Performance Data Mining: Automated Diagnosis, Adaption, and Optimization, vol. 18, no. 1, pp. 69–78, 2001.

[23] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," en, *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[24] F. E. Curtis, J. Huber, O. Schenk, and A. Wächter, "A note on the implementation of an interior-point algorithm for nonlinear optimization with inexact step computations," en, *Mathematical Programming*, vol. 136, no. 1, pp. 209–227, 2012.

[25] D. Kourounis, A. Fuchs, and O. Schenk, "Toward the Next Generation of Multiperiod Optimal Power Flow Solvers," *IEEE Transactions on Power Systems*, vol. 33, no. 4, pp. 4005–4014, 2018.

[26] Y. Wang, W. Yin, and J. Zeng, "Global Convergence of ADMM in Nonconvex Nonsmooth Optimization," en, *Journal of Scientific Computing*, vol. 78, no. 1, pp. 29–63, 2019.

[27] M. Hong, Z.-Q. Luo, and M. Razaviyayn, "Convergence Analysis of Alternating Direction Method of Multipliers for a Family of Nonconvex Problems," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 337–364, 2016.

[28] T. Lin, S. Ma, and S. Zhang, "On the global linear convergence of the ADMM with multiblock variables," *SIAM Journal on Optimization*, vol. 25, no. 3, pp. 1478–1497, 2015.

[29] B. He and X. Yuan, "On the $O(1/n)$ Convergence Rate of the Douglas–Rachford Alternating Direction Method," en, *SIAM Journal on Numerical Analysis*, vol. 50, no. 2, pp. 700–709, 2012.

[30] D. Davis and W. Yin, "Faster Convergence Rates of Relaxed Peaceman-Rachford and ADMM Under Regularity Assumptions," *Mathematics of Operations Research*, vol. 42, no. 3, pp. 783–805, 2017.

[31] B. Houska, J. Frasch, and M. Diehl, "An augmented Lagrangian based algorithm for distributed nonconvex optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1101–1127, 2016.

[32] A. Engelmann, Y. Jiang, T. Mühlpfordt, B. Houska, and T. Faulwasser, "Toward Distributed OPF Using ALADIN," *IEEE Transactions on Power Systems*, vol. 34, no. 1, pp. 584–594, 2019.

[33] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, New York, 2006.

[34] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," en, *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[35] M. W. Mehrez, T. Sprodowski, K. Worthmann, G. Mann, R. G. Gosine, J. K. Sagawa, and J. Pannek, "Occupancy grid based distributed MPC for mobile robots," in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE/RSJ, 2017, pp. 4842–4847.

[36] W. Hock and K. Schittkowski, "Test examples for nonlinear programming codes," *Journal of optimization theory and applications*, vol. 30, no. 1, pp. 127–129, 1980.

[37] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design, 2nd Edition*. Nob Hill Publishing, 2017.

[38] B. T. Stewart, S. J. Wright, and J. B. Rawlings, "Cooperative distributed model predictive control for nonlinear systems," *Journal of Process Control*, vol. 21, no. 5, pp. 698–704, 2011.

[39] M. A. Müller and F. Allgöwer, "Economic and Distributed Model Predictive Control: Recent Developments in Optimization-Based Control," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 2, pp. 39–52, 2017.

[40] X. Cai, M. Tippett, L. Xie, and J. Bao, "Fast distributed MPC based on active set method," *Computers & Chemical Engineering*, vol. 71, pp. 158–170, 2014.

[41] P. Christofides, J. Liu, and D. Munoz de la Pena, *Networked and distributed predictive control: Methods and nonlinear process network applications*. Springer Science & Business Media, 2011.

[42] D. P. Bertsekas, *Nonlinear programming*. Athena Scientific, Belmont, 1999.

[43] A. Engelmann, Y. Jiang, B. Houska, and T. Faulwasser, "Two decentralized algorithms for solving linear systems of equations from multi-agent optimization.," Karlsruhe Institute of Technology, Tech. Rep., 2020.