Alpaqa: Matrix-free solvers for nonlinear MPC

Pieter Pas & Alexander Bodard

October 20, 2023

STADIUS, KU Leuven

Overview

- Solving MPC problems using PANOC
- Solving MPC problems using ALPAQA
- Further enhancements to PANOC
- ALPAQA demo

Solving MPC problems using PANOC

Problem Statement: Nonlinear Optimal Control

$$
\begin{aligned}
\underset{x,u}{\textbf{minimize}} \quad & \sum_{k=0}^{N-1} \Big[\overbrace{\ell_k}^{\text{smooth stage cost}} (x^k, u^k) \Big] + \overbrace{\ell_N}^{\text{smooth stage cost}} (x^N) \\
\textbf{subject to} \quad & \underbrace{u^k \in U_k}_{\text{input constraints}}, && k \in \mathbb{N}_{[0,N-1]} \\
& \underbrace{x^{k+1} = f(x^k, u^k)}_{\text{dynamics}}, && k \in \mathbb{N}_{[0,N-1]} \\
& \underbrace{c_k(x^k) \in D_k}_{\text{state constraints}}, && k \in \mathbb{N}_{[0,N]}
\end{aligned}
$$

Problem Statement: Nonlinear Optimal Control

$$\underset{x,u}{\textbf{minimize}} \quad \sum_{k=0}^{N-1} \Big[\overbrace{\ell_k}^{\text{smooth stage cost}} (x^k, u^k) \Big] + \ell_N (x^N)$$

input constraints

$$\textbf{subject to} \quad u^k \in U_k, \qquad k \in \mathbb{N}_{[0,N-1]}$$

$$x^{k+1} = f(x^k, u^k), \qquad k \in \mathbb{N}_{[0,N-1]}$$

dynamics

$$c_k(x^k) \in D_k, \qquad k \in \mathbb{N}_{[0,N]}$$

Problem reformulation

$$\begin{aligned}
\underset{x,u}{\textbf{minimize}} \quad & \sum_{k=0}^{N-1} \left[\ell_k \left(x^k, u^k\right) \right] + \ell_N \left(x^N\right) \\
\textbf{subject to} \quad & u^k \in U_k, \qquad k \in \mathbb{N}_{[0,N-1]} \\
& x^{k+1} = f(x^k, u^k), \qquad k \in \mathbb{N}_{[0,N-1]}
\end{aligned}$$

$$\underset{u}{\textbf{minimize}} \quad \underbrace{\ell\left(\Phi(u; x^0), u\right)}_{\psi(u)} + \underbrace{\delta_U(u)}_{h(u)}$$

1. Eliminate dynamics $\rightarrow$ single-shooting formulation
2. Problem with smooth cost and box constraints $\rightarrow$ solve using PANOC

Proximal algorithms

$$\underset{u}{\textbf{minimize}} \quad \psi(u) + h(u)$$

Examples of h : indicators of boxes, probability simplex, semidefinite cone, second-order cones, regularizers (ℓ_1, ℓ_∞, nuclear norm)

How to solve optimization problems with nonsmooth cost?

- Indicator of convex set $\to$ Projected gradient method
- More general term $h \to$ Proximal gradient method

$$\mathbf{prox}_{\gamma h}(\bar{u}) \triangleq \arg\min_{x} \left(h(u) + \tfrac{1}{2\gamma} \|u - \bar{u}\|^2 \right)$$

Proximal operator

$$\mathbf{prox}_{\gamma h}(\bar{u}) \triangleq \operatorname*{arg\,min}_u \left(h(u) + \frac{1}{2\gamma} \|u - \bar{u}\|^2 \right)$$

Generalized projection

$$\begin{aligned} \mathbf{prox}_{\delta_C}(\bar{u}) &= \mathbf{proj}_C(\bar{u}) \\ &\triangleq \operatorname*{arg\,min}_{u \in C} \frac{1}{2} \|u - \bar{u}\|^2 \end{aligned}$$

Generalized gradient step

$$\begin{aligned} \mathbf{prox}_{\gamma \psi_{\mathrm{lin}}}(\bar{u}) &= \bar{u} - \gamma \nabla \psi(\bar{u}) \\ &= \operatorname*{arg\,min}_u \left(\psi(\bar{u}) + \nabla \psi(\bar{u})^\top (u - \bar{u}) + \frac{1}{2\gamma} \|u - \bar{u}\|^2 \right) \end{aligned}$$

PANOC: Solving the inner problem

$$\underset{u}{\mathbf{minimize}} \quad \psi(u) + h(u)$$

Proximal gradient iterations

Fixed-point iterations of

$$T_\gamma(u) \triangleq \mathbf{prox}_{\gamma h}\big(u - \gamma \nabla \psi(u)\big)$$

Guaranteed global convergence

Newton-type directions

Rootfinding of residual

$$R_\gamma(u) \triangleq \gamma^{-1}\big(u - T_\gamma(u)\big)$$

Fast local convergence

Line search on forward-backward envelope

PANOC

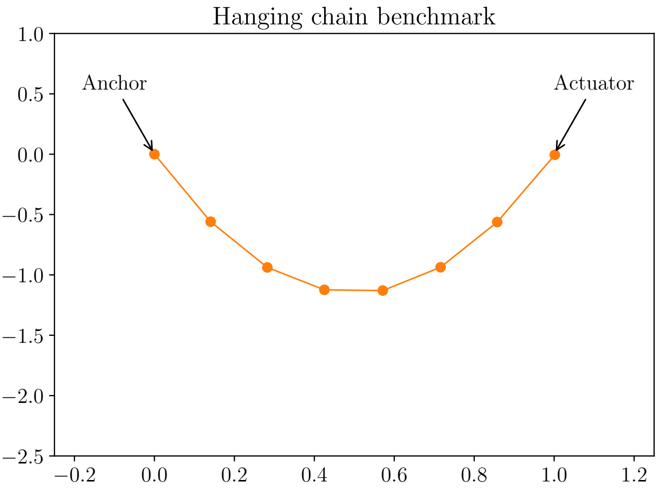
(Proximal Averaged Newton-type method for Optimality Conditions)

Lorenzo Stella et al. (2017). "A simple and efficient algorithm for nonlinear model predictive control". In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 1939–1944. DOI: 10.1109/CDC.2017.8263933

PANOC: Advantages

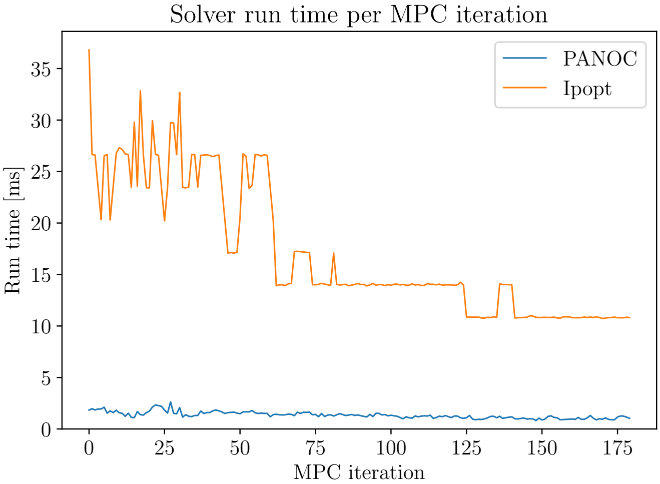
- \+ Supports nonconvex problems (Nonlinear dynamics)
- \+ First-order (Functions, gradients and proxes)
- \+ Matrix free (Using quasi-Newton directions)
- \+ Warm starting (Useful in MPC)
- – Requires proximable h (E.g. box constraints, but not general inequality constraints)
- – Selecting a suitable step size γ can be tricky (Numerical issues may arise)
- – Quasi-Newton directions may not be the best choice

MPC with input bounds

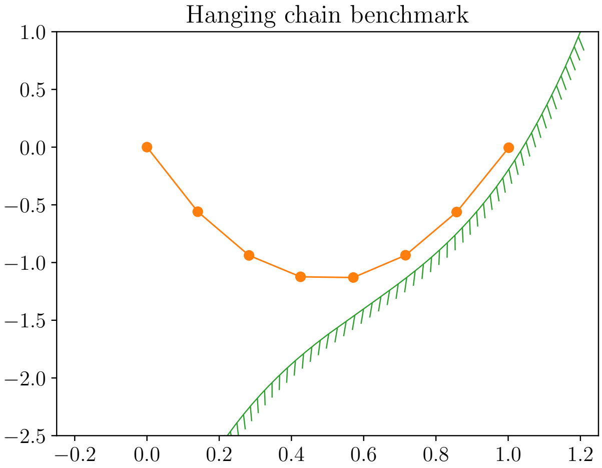


MPC with input bounds

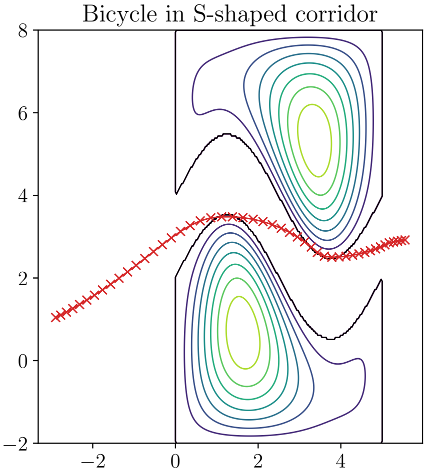
PANOC works well for MPC with box constraints on the inputs



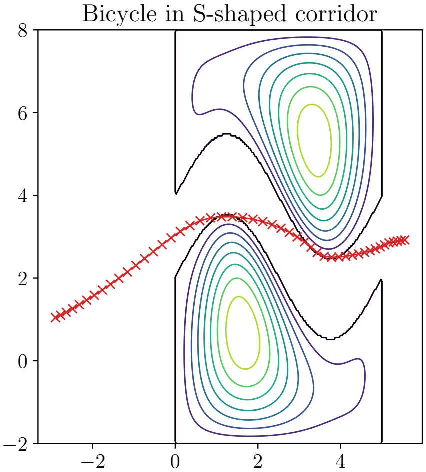
Limitations of PANOC



Limitations of PANOC



Limitations of PANOC



How to handle more **general** constraints?

→ Proximal operator would require projections onto complicated sets

Solving MPC problems using

ALPAQA

Solve general nonlinear programs of the form

smooth cost

$$\begin{aligned} \underset{x}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & x \in C \\ & g(x) \in D \end{aligned}$$

box constraints

general constraints

Pieter Pas, Mathijs Schuurmans, et al. (2022). "Alpaqa: A matrix-free solver for nonlinear MPC and large-scale nonconvex optimization". In: *2022 European Control Conference (ECC)*, pp. 417–422

Problem reformulation

$$\begin{aligned} \underset{x,u}{\textbf{minimize}} \quad & \sum_{k=0}^{N-1} \left[\ell_k \left(x^k, u^k\right) \right] + \ell_N \left(x^N\right) \\ \textbf{subject to} \quad & u^k \in U_k, && k \in \mathbb{N}_{[0,N-1]} \\ & c_k(x^k) \in D_k, && k \in \mathbb{N}_{[0,N]} \\ & x^{k+1} = f(x^k, u^k), && k \in \mathbb{N}_{[0,N-1]} \end{aligned} \quad \Rightarrow \quad \begin{aligned} \underset{x}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & x \in C \\ & g(x) \in D \end{aligned}$$

1. Eliminate dynamics $\rightarrow$ single-shooting formulation
2. Relax state constraints using augmented Lagrangian method
3. Solve inner problems with smooth cost and box constraints using PANOC or PANTR

PANOC as an inner solver

ALM requires solution of subproblems of the form

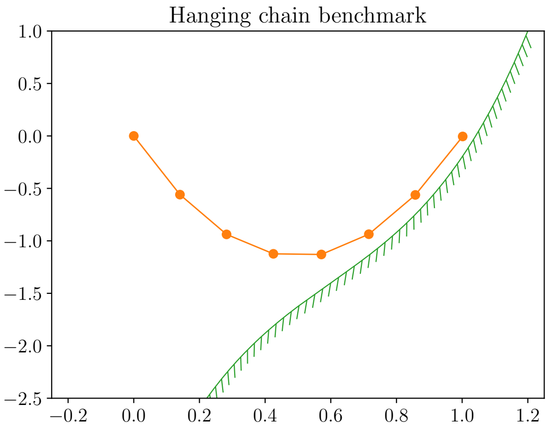
$$\underset{x}{\textbf{minimize}} \quad \psi(x) + h(x)$$

$\rightarrow$ Solve using PANOC

Constraint handling in ALPAQA:

- ‘Simple’ constraints $\rightarrow$ Directly with PANOC
- ‘Difficult’ constraints $\rightarrow$ ALM + PANOC

ALM + PANOC example



https://kul-optec.github.io/alpaqa/develop/Sphinx/examples/mpc/hanging-chain.html

Further enhancements

How to compute fast directions, quickly

Newton-type directions

- L-BFGS (Standard PANOC)
- Anderson
- Gauss-Newton (Useful in optimal control)
- Regularized Newton (Second-order information)

Exploitation of structure

- Structured PANOC (Active/inactive constraints, smaller systems)
- Dynamic programming (Fast solution of Gauss–Newton systems in OCPs)

Pieter Pas, Andreas Themelis, et al. (2022). "Gauss-Newton meets PANOC: A fast and globally convergent algorithm for nonlinear optimal control". In: *arXiv:2212.04391 [math.OC]*

Line-search methods

- First select update direction, then the step size
- Iterations are typically cheap (e.g. L-BFGS)
- Often works well

Trust-region methods

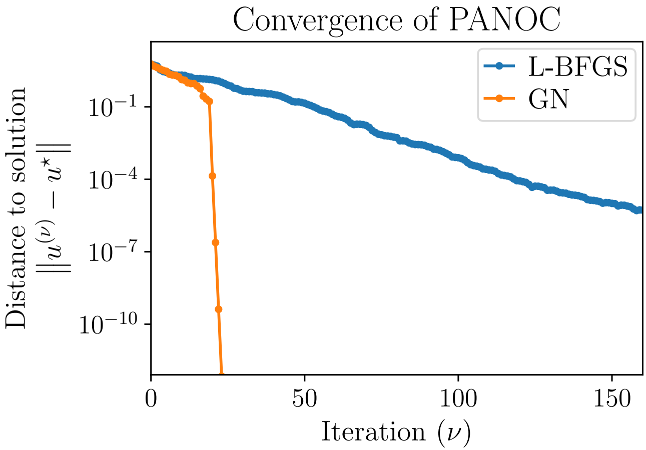
- Select update direction and step size simultaneously
- Iterations are considerably more expensive (Full subproblem per iteration)
- Better able to exploit curvature

$\rightarrow$ PANTR accelerates proximal-gradient iterations using trust-region strategies

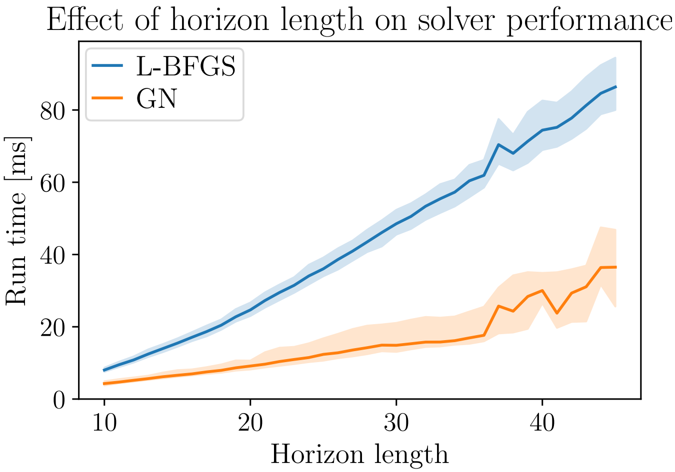
---

Alexander Bodard et al. (2023). "PANTR: A Proximal Algorithm With Trust-Region Updates for Nonconvex Constrained Optimization". In: *IEEE Control Systems Letters* 7, pp. 2389–2394. DOI: 10.1109/LCSYS.2023.3286331

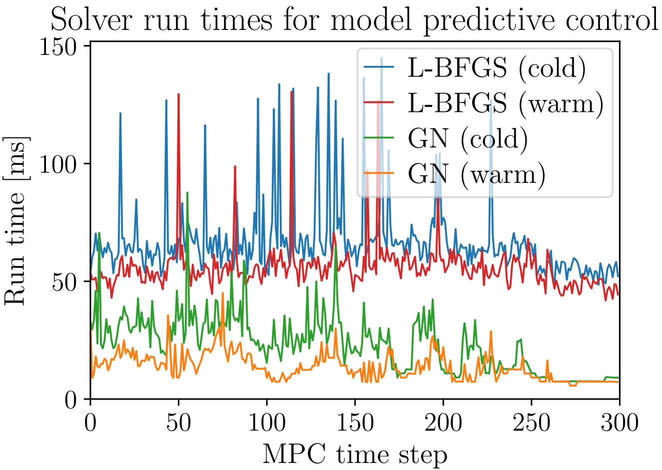
Results – PANOC with Gauss–Newton



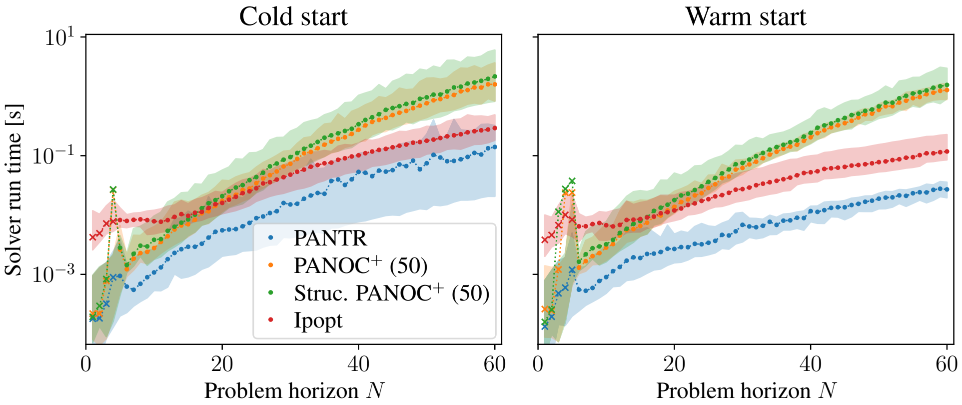
Results – PANOC with Gauss–Newton



Results – PANOC with Gauss–Newton



Results – PANTR



ALPAQA demo

The ALPAQA library

- Software package to solve NLPs of the form

$$\begin{aligned} \underset{x}{\textbf{minimize}} \quad & f(x) \\ \textbf{subject to} \quad & x \in C \\ & g(x) \in D \end{aligned}$$

- Efficient C++ implementations of PANOC, PANTR and similar algorithms, combined with augmented Lagrangian method for general constraints
- Easy-to-use **Python** interface
- Compatible with problems expressed in **CasADi**
- Open-source (LGPL)

Using ALPAQA

Installation

```
pip install --upgrade --pre alpaqa
```

Usage

1. Define **objective** and **constraint** functions (e.g. using CasADi)
2. Construct the minimization problem using `alpaqa.minimize`
3. Select one of the **solvers**
4. Pass the problem to the solver to get a **solution**
5. Tweak the solver **parameters** for optimal performance

ALPAQA example: 1. Problem definition

```python
# %% Build the problem (CasADi code, independent of alpaqa)
import casadi as cs

# Make symbolic decision variables
x1, x2 = cs.SX.sym("x1"), cs.SX.sym("x2")
x = cs.vertcat(x1, x2)  # Collect decision variables into one vector
# Make a parameter symbol
p = cs.SX.sym("p")

# Objective function f and the constraints function g
f = (1 - x1) ** 2 + p * (x2 - x1**2) ** 2
g = cs.vertcat(
    (x1 - 0.5) ** 3 - x2 + 1,
    x1 + x2 - 1.5,
)

# Define the bounds
C = [-0.25, -0.5], [1.5, 2.5]  # -0.25 <= x1 <= 1.5, -0.5 <= x2 <= 2.5
D = [-cs.inf, -cs.inf], [0, 0]  #         g1 <= 0,           g2 <= 0
```

$$
\begin{aligned}
\underset{x_1, x_2}{\textbf{minimize}} \quad & (1 - x_1)^2 + p\,(x_2 - x_1^2)^2 \\
\textbf{subject to} \quad & -0.25 \le x_1 \le 1.5 \\
& -0.5 \le x_2 \le 2.5 \\
& (x_1 - 0.5)^3 - x_2 + 1 \le 0 \\
& x_1 + x_2 - 1.5 \le 0
\end{aligned}
$$

https://kul-optec.github.io/alpaqa/develop/Sphinx/usage/getting-started.html

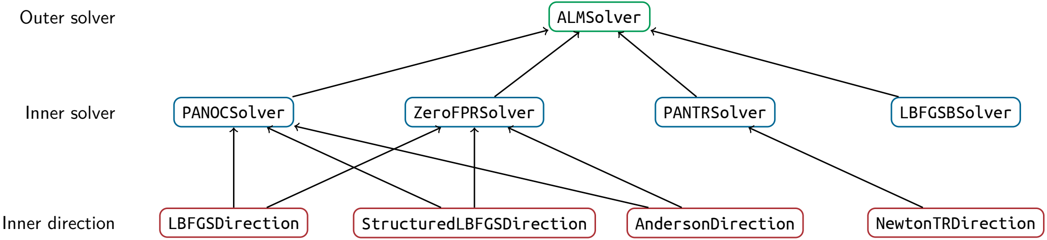
ALPAQA example: 2. alpaqa.minimize

```python
# %% Generate and compile C-code for the objective and constraints using alpaqa
from alpaqa import minimize

problem = (
    minimize(f, x)  #       Objective function f(x)
    .subject_to_box(C)  #   Box constraints x ∊ C
    .subject_to(g, D)  #    General ALM constraints g(x) ∊ D
    .with_param(p, [1])  #  Parameter with default value (can be changed later)
).compile()

# You can change the bounds and parameters after loading the problem
problem.param = [10.0]
problem.D.lowerbound[1] = -1e20
```

ALPAQA example: 3. Solver selection

```python
# %% Build a solver with the default parameters
import alpaqa as pa

inner_solver = pa.PANOCSolver()
solver = pa.ALMSolver(inner_solver)
```

ALPAQA example: 3. Solver selection



ALPAQA example: 3. Solver selection (with custom options)

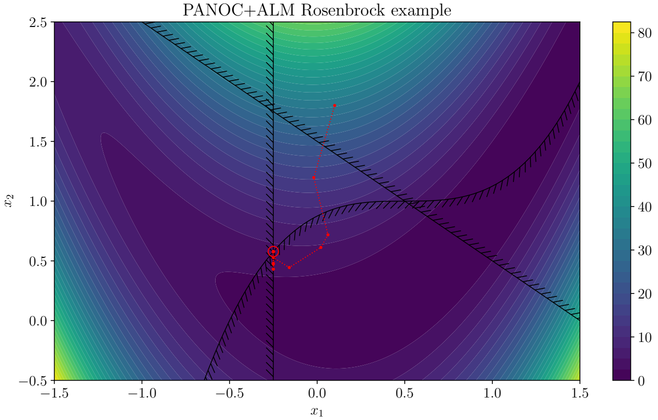
```python
# %% Build a solver with alternative fast directions and custom parameters

direction = pa.LBFGSDirection({'memory': 10})
inner_solver = pa.PANOCSolver(
    {
        "stop_crit": pa.FPRNorm,
        'print_interval': 1,
    },
    direction,
)
solver = pa.ALMSolver(
    {
        'tolerance': 1e-10,
        'dual_tolerance': 1e-10,
        'initial_penalty': 50,
        'penalty_update_factor': 20,
        'print_interval': 1,
    },
    inner_solver,
)
```

ALPAQA example: 4. Problem solution

```
# %% Compute a solution

x_sol, y_sol, stats = solver(problem)
```

- Solution vector x_{sol}
- Vector of Lagrange multipliers y_{sol}
- Dictionary of solver statistics

```python
# %% Compute a solution

x_sol, y_sol, stats = solver(problem)
```



ALPAQA example: 4. Problem solution (with initial guess)

```python
# %% Compute a solution

x_sol, y_sol, stats = solver(problem)

# %% Compute a solution starting with an initial guess

# Set initial guesses at arbitrary values
x0 = [0.1, 1.8]  # decision variables
y0 = [0.0, 0.0]  # Lagrange multipliers for g(x)

# Solve the problem
x_sol, y_sol, stats = solver(problem, x0, y0)

# Print the results
print(stats["status"])
print(f"Solution:      {x_sol}")
print(f"Multipliers:   {y_sol}")
print(f"Cost:          {problem.eval_f(x_sol):.5f}")
```

ALPAQA example: 5. Parameter tuning

How to get feedback?

→ Set `print_interval=1` for the ALM and inner solvers

```
┌─[PANOC]
│   φγ = +1.9384e+01,    ψ = +3.6851e+01, ‖∇ψ‖ = +5.6882e+01,  ‖p‖ = +6.1416e-01,    γ = +1.0797e-02,    ε = +2.3943e+01
├─      1
│   φγ = +1.2269e+01,    ψ = +1.5371e+01, ‖∇ψ‖ = +2.3969e+01,  ‖p‖ = +2.5880e-01,    γ = +1.0797e-02,    ε = +1.8779e+01
│  ‖q‖ = +4.8637e-01,    τ = +1.000e+00,       dir update accepted
├─      2
│   φγ = +6.8327e+00,    ψ = +6.9594e+00, ‖∇ψ‖ = +4.8435e+00,  ‖p‖ = +5.2296e-02,    γ = +1.0797e-02,    ε = +4.3872e+00
│  ‖q‖ = +1.1310e-01,    τ = +1.000e+00,       dir update accepted
├─      3
│   φγ = +6.3489e+00,    ψ = +6.6338e+00, ‖∇ψ‖ = +7.2651e+00,  ‖p‖ = +7.8442e-02,    γ = +1.0797e-02,    ε = +8.4532e+00
│  ‖q‖ = +2.4668e-01,    τ = +1.000e+00,       dir update accepted
├─      4
│   φγ = +3.9806e+00,    ψ = +4.9130e+00, ‖∇ψ‖ = +1.8617e+01,  ‖p‖ = +9.4935e-02,    γ = +5.3986e-03,    ε = +9.4964e+00
│  ‖q‖ = +2.6553e-01,    τ = +2.500e-01,       dir update accepted
├─      5
│   φγ = +3.6639e+00,    ψ = +3.7868e+00, ‖∇ψ‖ = +9.3303e+00,  ‖p‖ = +3.6430e-02,    γ = +5.3986e-03,    ε = +4.1981e+00
│  ‖q‖ = +4.7297e-02,    τ = +1.000e+00,       dir update accepted
├─      6
│   φγ = +3.5141e+00,    ψ = +3.5460e+00, ‖∇ψ‖ = +1.0537e+01,  ‖p‖ = +1.8557e-02,    γ = +5.3986e-03,    ε = +2.1384e+00
│  ‖q‖ = +4.9105e-02,    τ = +1.000e+00,       dir update accepted
├─      7
│   φγ = +3.4616e+00,    ψ = +3.4616e+00, ‖∇ψ‖ = +1.3613e+01,  ‖p‖ = +1.4385e-17,    γ = +5.3986e-03,    ε = +2.6645e-15
└─ Converged ──
[ALM]        0: ‖Σ‖ = +7.07106781e+01, ‖y‖ = +7.36607143e+00, δ = +1.47321429e-01, ε = +2.66453526e-15, status =     Converged, iter =             7
┌─[PANOC]
│   φγ = +1.1790e+01,    ψ = +1.4883e+01, ‖∇ψ‖ = +3.0077e+02,  ‖p‖ = +4.1992e-02,    γ = +2.8504e-04,    ε = +1.0449e+02
├─      1
│   φγ = +8.0401e+00,    ψ = +9.5961e+00, ‖∇ψ‖ = +2.1840e+02,  ‖p‖ = +2.9783e-02,    γ = +2.8504e-04,    ε = +7.4111e+01
│  ‖q‖ = +1.0244e-01,    τ = +1.000e+00,       dir update accepted
├─      2
│   φγ = +4.2441e+00,    ψ = +4.2441e+00, ‖∇ψ‖ = +1.9932e+01,  ‖p‖ = +3.4936e-17,    γ = +2.8504e-04,    ε = +1.2257e-13
└─ Converged ──
[ALM]        1: ‖Σ‖ = +1.00124922e+03, ‖y‖ = +1.02547269e+01, δ = +2.88865546e-03, ε = +1.22568622e-13, status =     Converged, iter =             2
```

ALPAQA example: 5. Parameter tuning

Useful parameters

- `LBFGSParams.memory`: the length of the history to keep for the L-BFGS direction (too low: poor approximation of Newton direction, too high: too reliant on old iterates; decrease if `stats["inner"]["linesearch_backtracks"]` is too high)
- `ALMParams.initial_penalty`: penalty parameter value for the first inner problem (lower: easier subproblem, higher: better constraint satisfaction)
- `ALMParams.penalty_update_factor`: how aggressively the penalty factor is increased after each ALM iteration (too high: harder, ill-conditioned inner problems)
- `PANOCParams.max_iter`: maximum number of iterations when solving an inner problem (increase if you see many inner convergence failures in the `[ALM]` output)

ALPAQA example: 5. Parameter tuning

Tolerances and stopping criteria

- `PANOCParams.stop_crit`: determines residual function and norm to use when checking for termination (e.g. fixed-point residual or projected gradient norm)
- `ALMParams.tolerance`: primal tolerance, Lagrangian stationarity or similar (specific formula depends on `PANOCParams.stop_crit`)
- `ALMParams.dual_tolerance`: combination of constraint violation and complementary slackness

Installation

`pip install --upgrade --pre alpaqa`

Source code and issue tracker

`https://github.com/kul-optec/alpaqa`

Documentation and examples

`https://kul-optec.github.io/alpaqa/develop/Sphinx/index.html`

Contact

`pieter.p.dev@outlook.com`