SOLUTION METHODS FOR GENERALIZED NASH EQUILIBRIUM PROBLEMS AND GAME-THEORETIC CONTROL

Alberto Bemporad

imt.lu/ab

Funded by the European Union

European Research Council
Established by the European Commission

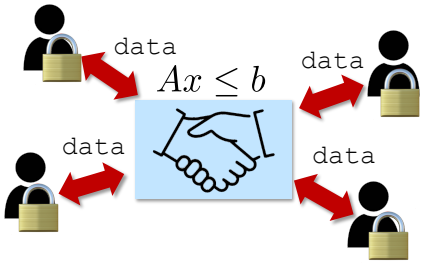
IMT SCHOOL FOR ADVANCED STUDIES LUCCA

Game on! Seminars - KTH - March 10, 2026

CONTENTS OF MY TALK

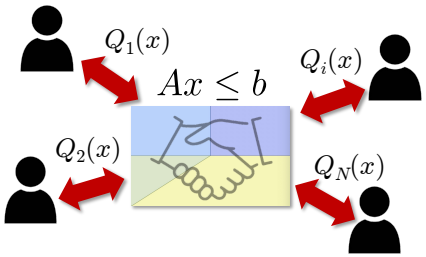
1. An **active learning** method to find generalized Nash equilibria (GNEs) from **best-response data**

```
pip install gnep-learn
```



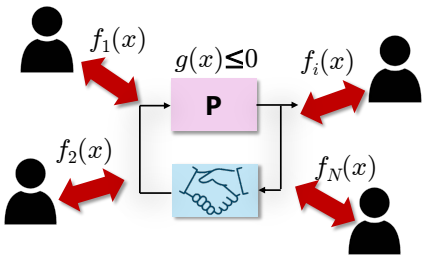
2. A **multiparametric solver** for **linear-quadratic** GNE problems

```
pip install nash-mpqp
```

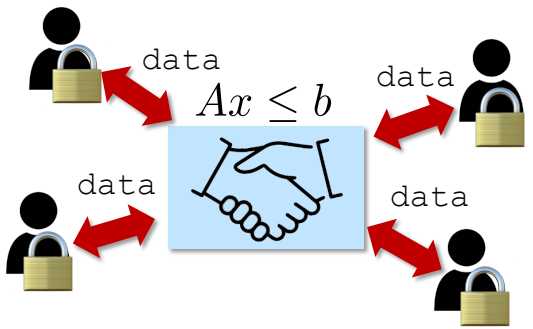


3. A general library for **computing** GNEs and solving **game-design** and **game-theoretic control** problems

```
pip install nashopt
```



LEARNING NASH EQUILIBRIA FROM DATA



Contents based on the following paper:

F. Fabiani and A. Bemporad, "*An active learning method for solving competitive multi-agent decision-making and control problems*," IEEE TAC, 2025

GNEP PROBLEM

- N (**non-cooperative**) agents, mutually interacting, each one solving

$$f_i(x_{-i}) = \underset{x_i \in \mathcal{X}_i}{\arg\min} \quad J_i(x_i, x_{-i}) \qquad x_i \in \mathbb{R}^{n_i}$$
$$\text{s.t.} \qquad (x_i, x_{-i}) \in \Omega \qquad x_{-i} \in \mathbb{R}^{n-n_i}$$

- **Objectives** J_i are **private**, **global constraint set** Ω is **shared**.
  Possible local constraints $x_i \in \mathcal{X}_i$ are either private or handled in Ω

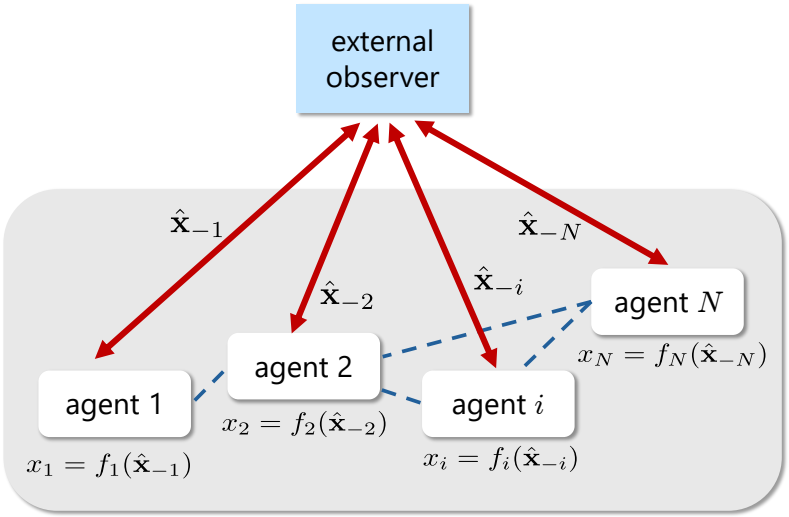
- **Best responses** $x_i = f_i(x_{-i})$ are single valued and can be **queried** at given x_{-i}

- **Goal**: learn a **generalized Nash equilibrium (GNE)** $x^\star = (x_1^\star, \dots, x_N^\star) \in \Omega$:

$$x_i^\star = f_i(x_{-i}^\star) \qquad \forall i = 1, \dots, N$$

ACTIVE-LEARNING APPROACH

- An **external observer** **queries** each agent i at selected $\hat{x}_{-i}$ and collect best responses $x_i = f_i(\hat{x}_{-i})$



- Use collected data to **update** an **affine surrogate model** $\hat{f}_i$ of best-response f_i

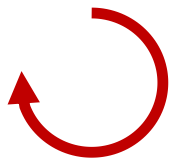
$$\hat{f}_i(x_{-i}, \theta_i) = \Gamma_i \begin{bmatrix} x_{-i} \\ 1 \end{bmatrix} \qquad \theta_i = \mathrm{vec}(\Gamma_i)$$

- **compute** an **approximate GNE**: **(constrained least-squares)**

$$\hat{x}^\star = \underset{x \in \Omega}{\mathrm{argmin}} \sum_{i=1}^{N} \left\| x_i - \hat{f}_i(x_{-i}, \theta_i) \right\|_2^2$$

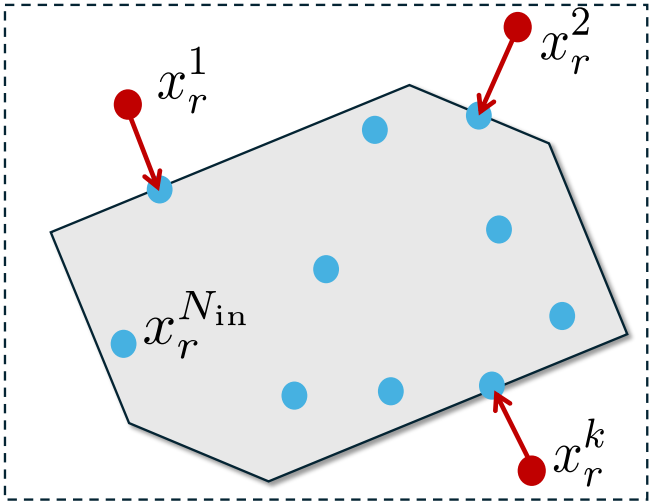
(minimum norm solution taken if multiple solutions exist)

- **Repeat** by querying each agent at $\hat{x}^\star_{-i}$ until convergence (if any occurs)



INITIALIZATION

- Select initial points x_r^k **randomly**, $k = 1, \ldots, N_{\text{in}}$ in a hyper-box of interest (e.g., random or Latin-hypercube sampling)



- **Project** them onto Ω:

$$\hat{x}^k = \underset{x \in \Omega}{\text{argmin}} \; \|x - x_r^k\|_2^2$$

- For each point $\hat{x}^k$, get each agent's **best response** $x_i^k = f_i(\hat{x}^k_{-i})$

- Collect **initial dataset** $\{(\hat{x}^k_{-i}, x_i^k), k = 1, \ldots, N_{\text{in}}\}$

RECURSIVE BR-MODEL LEARNING

- Use bank of **linear Kalman filters** to **estimate** model parameters θ_i **recursively**:

$$\phi_i^k = \begin{bmatrix} \hat{x}_{-i}^k \\ 1 \end{bmatrix}, \; a_i^k = P_i^k \phi_i^k$$

$$P_i^{k+1/2} = P_i^k - \frac{a_i^k (a_i^k)'}{1 + (\phi_i^k)' a_i^k}$$

$$\theta_i^{k+1} = \theta_i^k + P_i^{k+1/2} \phi_i^k (\mathbf{x_i^k} - (\phi_i^k)' \theta_i^k)$$

$$P_i^{k+1} = P_i^{k+1/2} + \beta I$$

$$\theta_i = \mathrm{vec}(\Gamma_i)$$

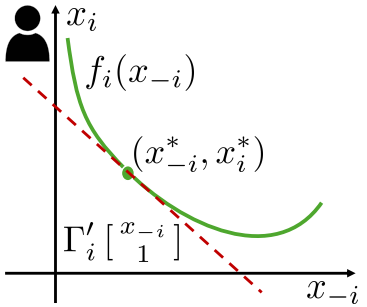
- β = **learning rate**. The larger β, the faster the estimator converges

- $P_0^i = \alpha I$. The smaller α, the larger the ℓ_2**-regularization** on θ^i

CONVERGENCE PROPERTIES

- **Main result**: If $\lim_{k\to\infty} \theta_i^k = \tilde{\theta}_i$, for all $i = 1, \ldots, N$ then $\lim_{k\to\infty} \hat{x}^k = x^\star$ and $x^\star$ is an equilibrium of the **original** GNEP, i.e., $x_i^\star = f_i(x_{-i}^\star)$

- Each affine surrogate $\hat{f}_i$ coincides with f_i **pointwise** at $x_{-i}^\star$



☹ **Weakness**: we cannot guarantee if $\lim_{k\to\infty} \theta_i^k$ exist for all i

☺ Strength: very **minimal assumptions**: f_i continuous, Ω nonempty
No other assumption!

SIMPLE NUMERICAL EXAMPLE

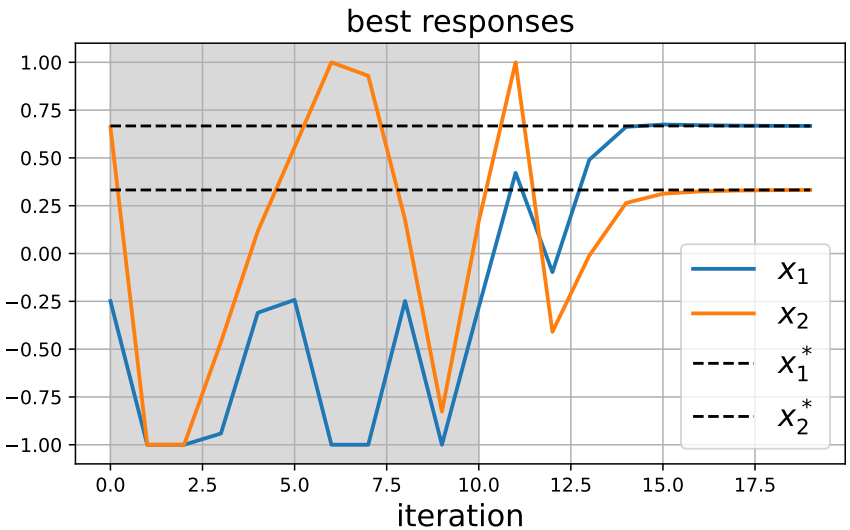
- Nash equilibrium problem:

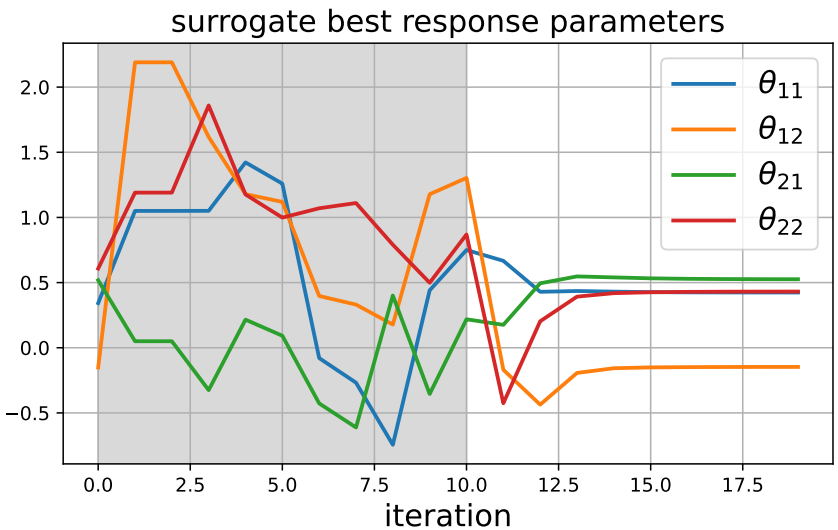
$$x_1^\star = \underset{-1\le x_1\le 1}{\operatorname{argmin}}\ x_1^4 - x_1x_2 - x_1 \quad \triangleq f_1(x_2)$$

$$x_2^\star = \underset{-1\le x_2\le 1}{\operatorname{argmin}}\ x_2^4 + x_1x_2 - x_2 \quad \triangleq f_2(x_1)$$

- Generate $N_{\text{in}} = 10$ initial random samples in $[-1,1]^2$ and run active-learning algorithm for another 10 iterations

- Result: $x_1^\star \approx 0.7077, x_2^\star = 0.4181$





PYTHON PACKAGE

gnep-learn

```python
from gnep_learn import GNEP

n = 2  # number of agents
dim = [1, 1]  # each agent optimizes one variable
N_init = 10 # number of initial random sampling iterations
N = 20 # total number of iterations

def J1(x1,x2):
    return x1**4 -x1*x2 - x1 # function minimized by agent #1
def J2(x2,x1):
    return x2**4 +x1*x2 - x2 # function minimized by agent #2

J = [lambda x1, x2: J1(x1,x2)[0],
     lambda x2, x1: J2(x2,x1)[0]]  # agents' objectives

lbx = -1. * np.ones(2) # lower bound on x
ubx = 1. * np.ones(2) # upper bound on x

gnep = GNEP(J, dim, lbx=lbx, ubx=ubx) # create GNEP object

xeq, XX, XXeq, _ = gnep.learn(N=N, N_init=N_init) # learn equilibrium
```

```
pip install gnep-learn
```

github.com/bemporad/gnep-learn

NUMERICAL EXAMPLE - NO NASH EQUILIBRIUM EXISTS

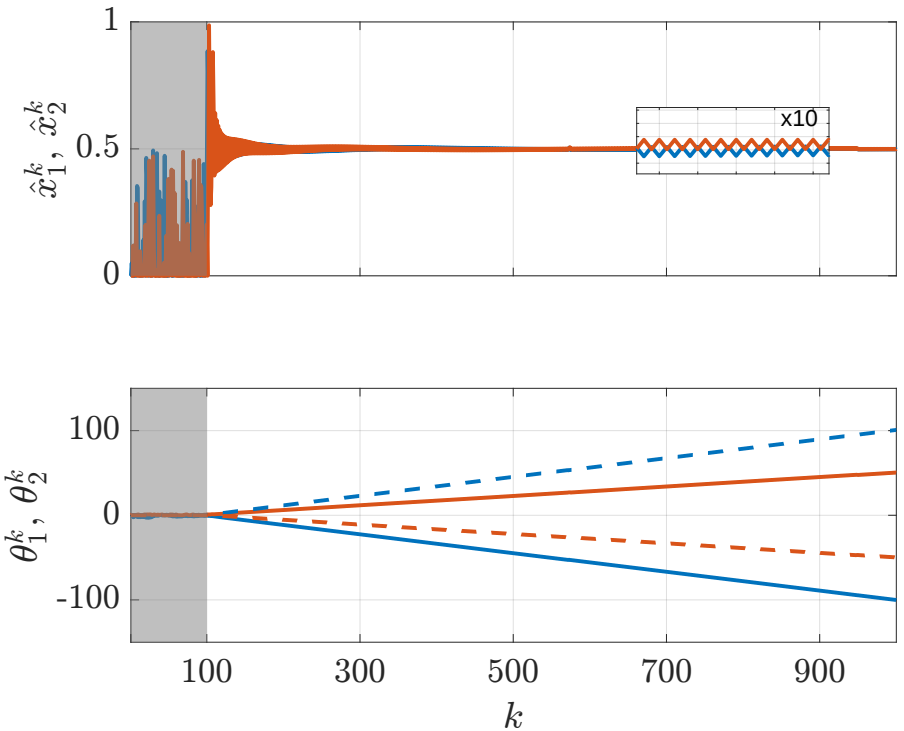
(Fabiani, Bemporad, 2025)

- Nash equilibrium problem with $N = 2$ agents, no equilibrium exists

$$f_1(x_2) = \underset{0 \le x_1 \le 1}{\operatorname{argmin}} \; -x_1^2 + 2x_1x_2$$

$$f_2(x_1) = \underset{0 \le x_2 \le 1}{\operatorname{argmin}} \; x_2 - 2x_1x_2$$

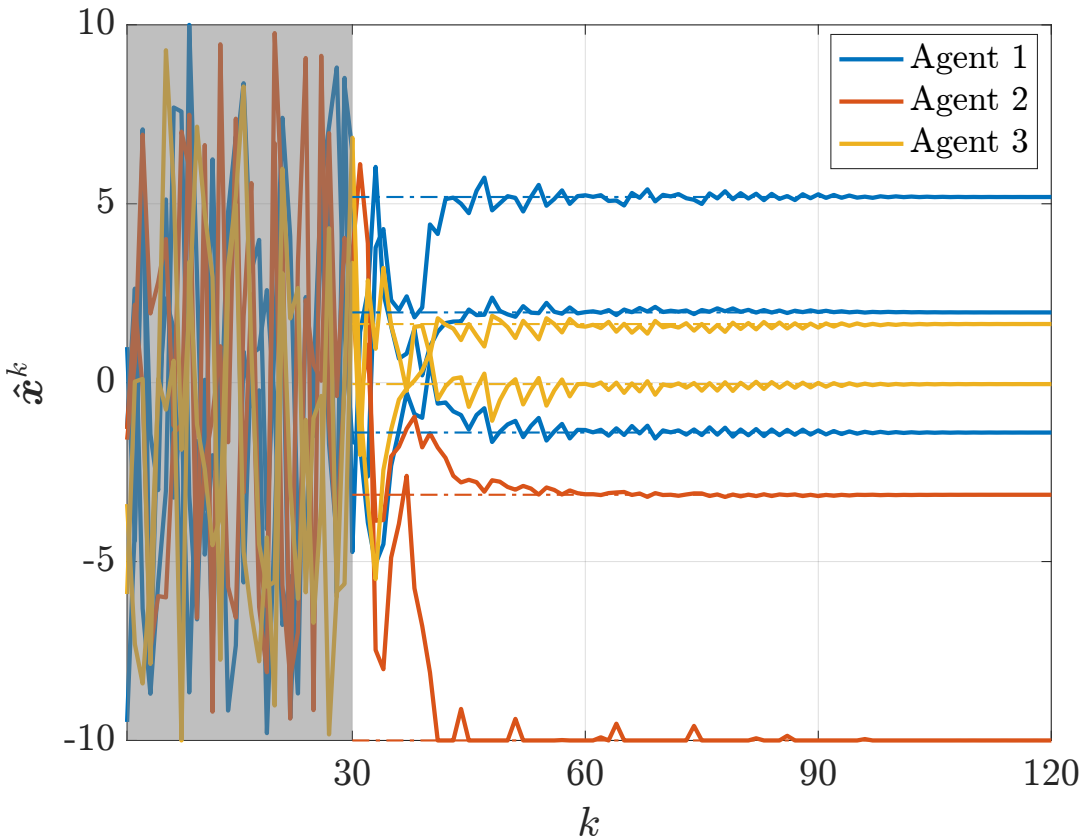
- Learning rate $\beta = 1$
- Best response **oscillates** around $\frac{1}{2}$
- Parameter vector θ **diverges**



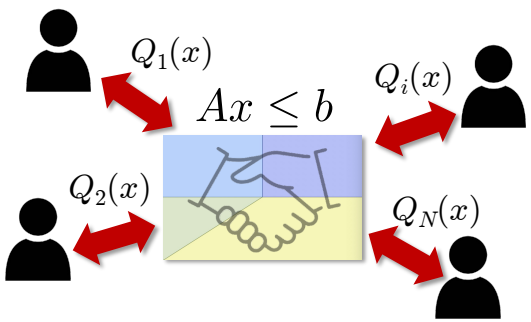
NUMERICAL EXAMPLE

(Facchinei, Kanzow, 2009 - Example 3)

- GNEP with $N = 3$ agents of dimensions $3, 2, 2$
- Coupling constraint: $\mathbf{1}'x \leq 1$
- $N_{\text{in}} = 30$ initial random samples, active-learning algorithm run for 90 iterations
- Learning rate $\beta = 1$



MULTIPARAMETRIC LINEAR QUADRATIC GNEP



Contents based on the following paper:

S. Hall and A. Bemporad, "*Solving Multiparametric Generalized Nash Equilibrium Problems and Explicit Game-Theoretic Model Predictive Control*, arXiv, 2025

MULTIPARAMETRIC LINEAR QUADRATIC GAMES

- N agents with best responses

$$\begin{cases} x_i^\star(x_{-i}, p) = \arg\min\limits_{x_i} \dfrac{1}{2} x' Q_i x + (c_i + F_i p)' x \\ \text{s.t. } Ax \le b + Sp \end{cases}$$

parameter $p \in \mathbb{R}^{n_p}$

- Best response = solution to a **multiparametric Quadratic Programming** (mpQP) problem with parameters (x_{-i}, p)
- Assume each QP #i is **strictly convex** wrt $x_i \Rightarrow$ best response is **unique**
- Possible **local constraints** on x_i embedded in global constraints $Ax \le b + Sp$

MULTIPARAMETRIC LINEAR QUADRATIC GAMES

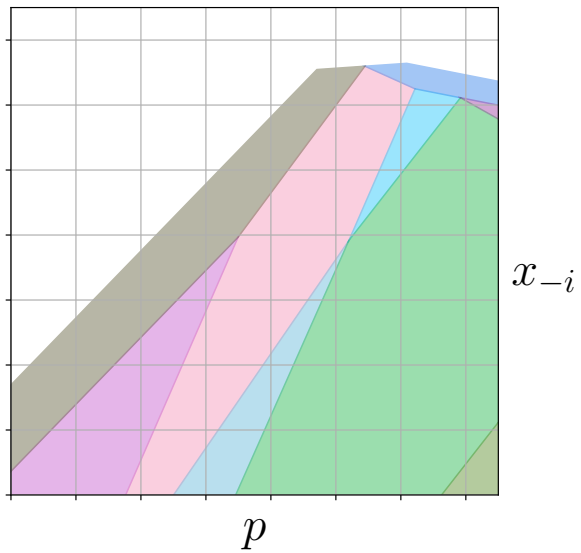
- The best response $x_i^\star(x_{-i}, p)$ is a **continuous piecewise affine** function of (x_{-i}, p) over a **polyhedral partition** of the (x_{-i}, p)-space

(Bemporad, Morari, Dua, Pistikopoulos, 2002)

$$x_i^\star(x_{-i}, p) = \begin{cases} H_{-i}^1 x_{-i} + G_i^1 p + h_i^1 & \text{if } (x_{-i}, p) \in CR_i^1 \\ \vdots & \vdots \\ H_{-i}^{N_i} x_{-i} + G_i^{N_i} p + h_i^{N_i} & \text{if } (x_{-i}, p) \in CR_i^{N_i} \end{cases}$$

$$CR_i^j = \left\{ (x_{-i}, p) : \; C_{-i}^j x_{-i} + D_i^j p \leq e_i^j \right\}$$

$$j = 1, \ldots, N_i$$



- Each critical region CR_i^j corresponds to an optimal active set of QP #i
- We have N partitions, one per agent i, with N_i critical regions (CRs)

EQUILIBRIUM CONDITIONS

- Consider a **combination** $\mathcal{C}_k = (j_1, \ldots, j_N)$ of agents' CRs, $k = 1, \ldots, \prod_{i=1}^{N} N_i$

- A combination $\mathcal{C}_k$ is **valid** if, for each constraint involving agents i and j, either both have it active or both inactive

- **Equilibrium condition** for $\mathcal{C}_k = (j_1, \ldots, j_N)$:

$$\begin{cases} x_1 = H_{-1}^{j_1} x_{-1} + G_1^{j_1} p + h_1^{j_1} \\ \vdots \quad\quad \vdots \\ x_N = H_{-N}^{j_N} x_{-N} + G_N^{j_N} p + h_N^{j_N} \end{cases} \quad \text{and} \quad \begin{cases} C_{-1}^{j_1} x_{-1} + D_1^{j_1} p \le e_1^{j_1} \\ \vdots \\ C_{-N}^{j_N} x_{-N} + D_N^{j_N} p \le e_N^{j_N} \end{cases}$$

- A **critical region** for the **mp-GNEP** associated with $\mathcal{C}_k$ is the projection onto the p-space of the polyhedron defined by the above constraints

CRITICAL REGIONS WITH UNIQUE EQUILIBRIUM

- For each valid combination $\mathcal{C}_k$, rewrite the equilibrium condition as

$$M_x^k x = M_p^k p + M_1^k \qquad M_x^k \in \mathbb{R}^{n\times n} \qquad \text{(parametric linear system)}$$

- If M_x^k is invertible, we get a **unique equilibrium** for all p:

$$x^\star(p) = (M_x^k)^{-1} M_p^k p + (M_x^k)^{-1} M_1^k$$

with associated **polyhedral** critical region

$$CR_k = \left\{ p : C_{-i}^{j_i} x^\star(p) + D_i^{j_i} p \le e_i^{j_i}, \quad i = 1, \dots, N \right\}$$

- Only nonempty full-dimensional CR's are stored (based on Chebyshev radius)

- **Note**: CR_k's may **overlap** ⇒ **multiple equilibria** exist for the same p

NO EQUILIBRIUM AND MULTIPLE EQUILIBRIA

- If M_x^k is singular, compute **SVD decomposition** $M_x^k = U\Sigma V'$

- Change variables $y_1 = V_1'x, y_2 = V_2'x$ to rewrite the equilibrium condition as

$$\begin{aligned} y_1 &= \Sigma_1^{-1}(U_1'M_p^k p + U_1'M_1^k) \\ 0 &= U_2'M_p^k p + U_2'M_1^k \end{aligned} \qquad U = [U_1\, U_2], \quad V = [V_1\, V_2]$$

- Potential (**infinitely-many**) Nash equilibria:

$$x^\star(p) = V_1\Sigma^{-1}(U_1'M_p^k p + U_1'M_1^k) + V_2 y_2 \qquad y_2 = \text{free vector}$$

- Conditions for existence of **full-dimensional** critical regions and equilibria:

$$U_2'M_p^k = 0,\ U_2'M_1^k = 0$$

and the **projection** $CR_k = \left\{p : \exists y_2 : C_{-i}^{j_i} x^\star(p) + D_i^{j_i} p \le e_i^{j_i},\ i = 1, \dots, N\right\}$ is nonempty and full-dimensional

CHOOSING AMONG MULTIPLE EQUILIBRIA

- In CR_k's with ∞-many equilibria, we can select the **minimum-norm solution**:

$$\|x^\star(p, y_2)\|_2^2 = \|\Sigma^{-1}U_1'(M_p^k p + U_1' M_1^k)\|_2^2 + \|y_2\|_2^2$$

- We can split CR_k into **sub-regions** by solving the following mpQP for $p \in CR_k$:

$$\begin{aligned} y_2^\star(p) = \arg\min\ & \|y_2\|_2^2 \\ \text{s.t. } & C_{-i}^{j_i} x^\star(p, y_2) + D_i^{j_i} p \le e_i^{j_i}, \quad i = 1, \dots, N \end{aligned}$$

- **Alternative**: given a **social welfare cost** $f(x)$, solve instead the mpQP

$$\begin{aligned} y_2^\star(p) = \arg\min\ & f(x^\star(p, y_2)) \\ \text{s.t. } & C_{-i}^{j_i} x^\star(p, y_2) + D_i^{j_i} p \le e_i^{j_i} \\ & i = 1, \dots, N \end{aligned}$$

e.g.: $f(x)$ **utilitarian sum** of agents' objectives

$$f(x) = \sum_{i=1}^{N} \frac{1}{2} x' Q_i x + (c_i + F_i p)' x$$

- In both cases, set the unique equilibrium $x^\star(p) = x^\star(p, y_2^\star(p))$ (PWA function)

VARIATIONAL GNES

- A **variational GNE** is an equilibrium where all the agents involved in a **shared constraint** have equal **dual variables** (=cost sensitivity for that constraint)

- From the **KKT conditions** of each agent's mpQP, we retrieve the **dual variables** $\lambda_i^\star(x_{-i},p) = \Lambda_x x_{-i} + \Lambda_p p + \lambda_1$ associated with coupling constraints

- To compute a variational GNE:
  - for each considered constraint c involving at least two agents, **add**

$$\Lambda_x^j x_{-j} + \Lambda_p^j p + \lambda_1^j = \Lambda_x^i x_{-i} + \Lambda_p^i p + \lambda_1^i, \quad \forall j \neq i$$

    to the equilibrium conditions $M_x^k x = M_p^k p + M_1^k$

  - recompute SVD and check if vGNE solutions $x^\star(p)$ exist

  - compute (sub)region $\overline{CR}_k \subseteq CR_k$ as before

PYTHON PACKAGE

```
from nash_mpqp import NashMPQP

dim = [2,3,2]  # number of variables for each agent
split = "variational" # or None, "min-norm", "welfare", "variational-split"

nash_mpqp = NashMPQP(dim, pmin, pmax, xmin, xmax,
                     Q, c, F, A, b, S, lb, ub, split = split)

nash_mpqp.solve()
```

`lb`, `ub`: bounds on variables (finite or $\pm\infty$)

`pmin`, `pmax`: range of parameters

`xmin`, `xmax`: range of variables

`pip install nash-mpqp`

`github.com/bemporad/nash_mpqp`

EXAMPLE: SIMPLE 2-AGENT GNEP

- 2 agents, each controlling one variable x_1, x_2 (Rosen, 1965)

$$\min_{x_1 \geq 0} \frac{1}{2}(x_1)^2 - x_1 x_2 + p_2 x_1$$
$$\text{s.t.} -x_1 - x_2 \leq p_1$$

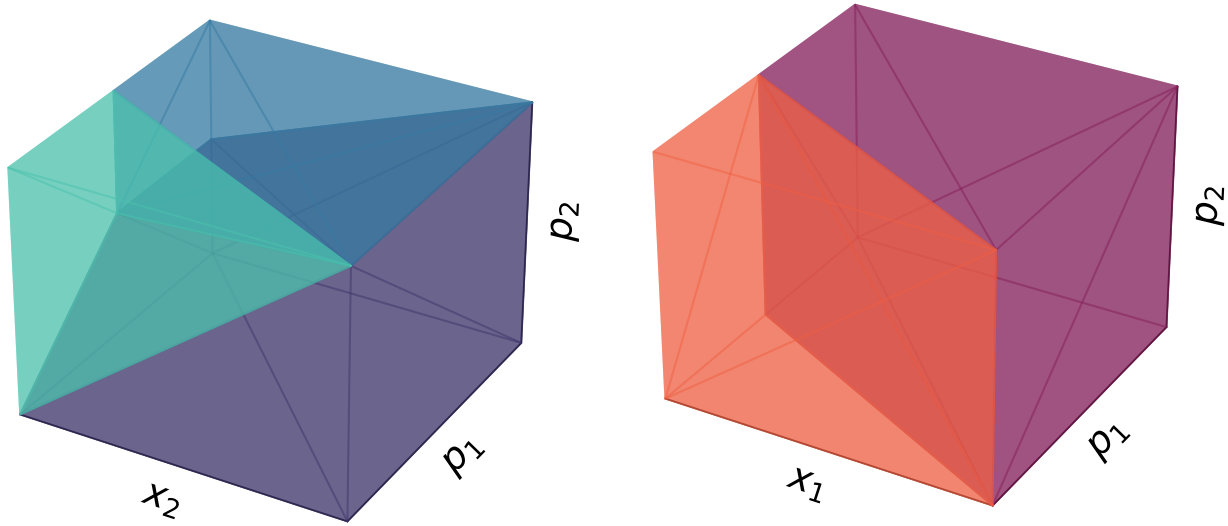
$$Q_1 = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}, \; F_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\min_{x_2 \geq 0} (x_2)^2 + x_1 x_2$$
$$\text{s.t.} -x_1 - x_2 \leq p_1$$

$$Q_2 = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$$

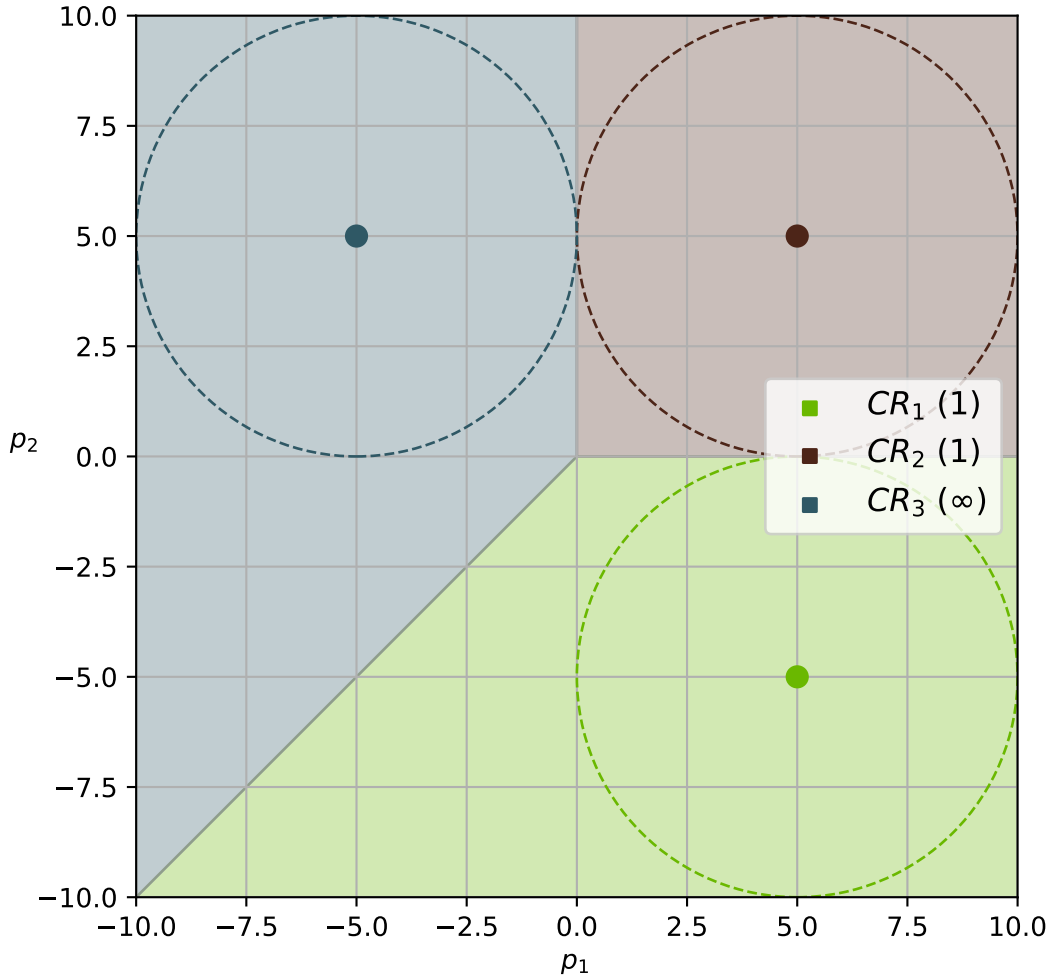
- Parameter vector: $p = [p_1 \; p_2]'$

- CRs of each agent's best-response mpQP:



EXAMPLE: SIMPLE 2-AGENT GNEP

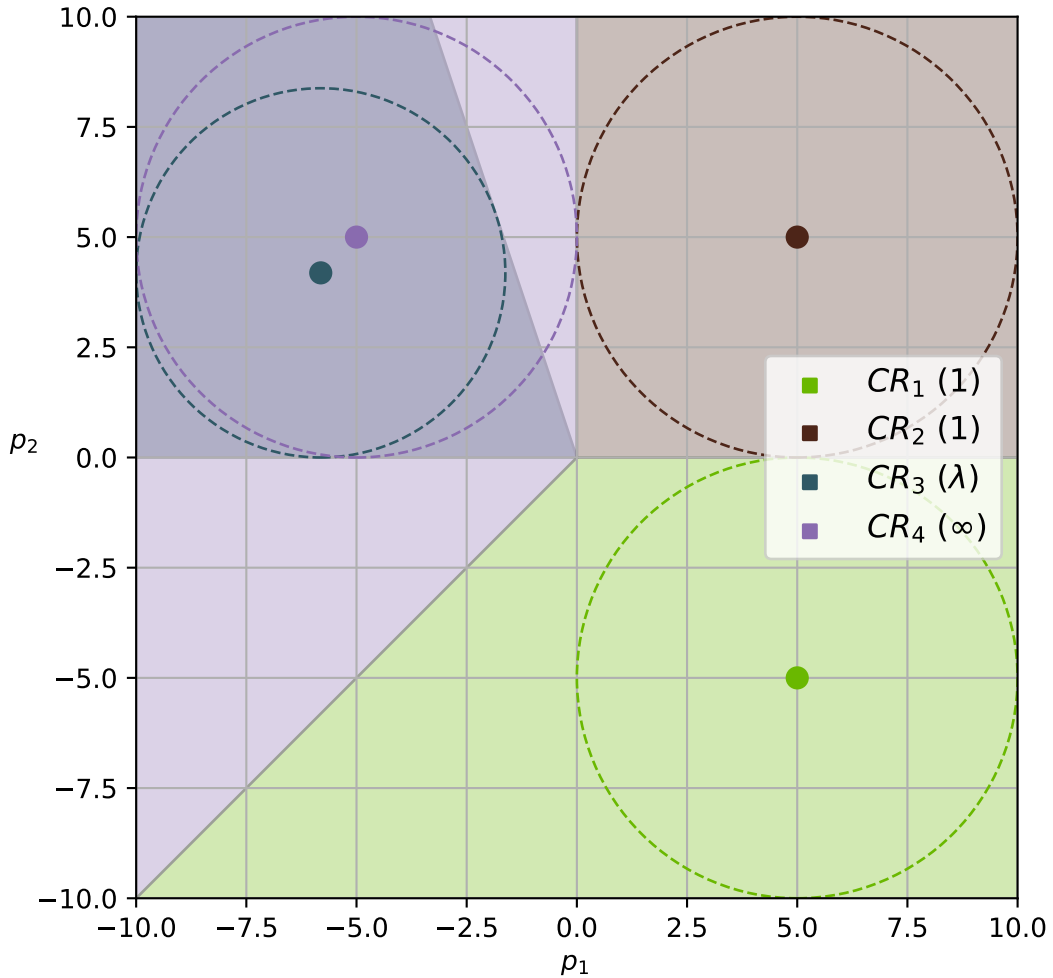
- **Critical regions:** no splitting in the case of infinitely-many equilibria



(1=single solution, ∞ = infinitely-many solutions)

EXAMPLE: SIMPLE 2-AGENT GNEP

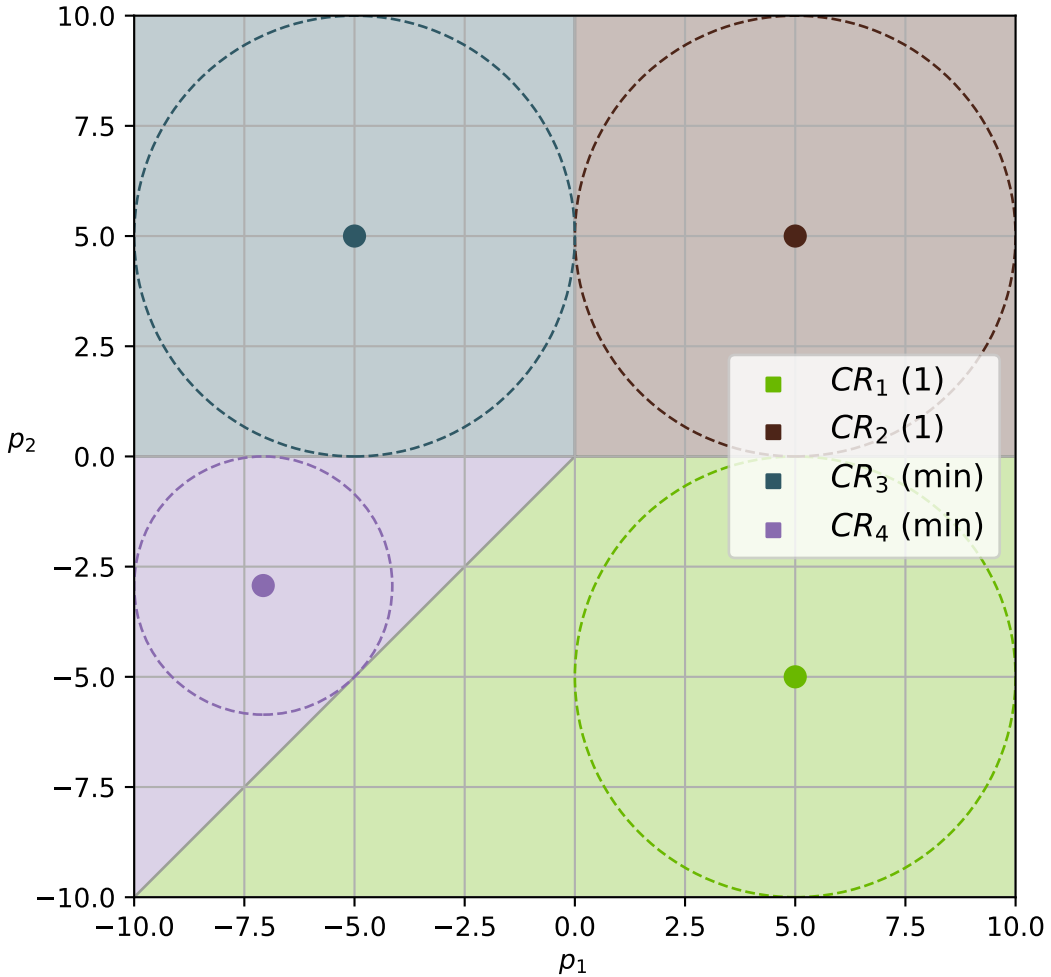
- **Critical regions**: adding **variational-GNE** regions



(1=single solution, ∞ = infinitely-many solutions, λ = variational GNE)

EXAMPLE: SIMPLE 2-AGENT GNEP

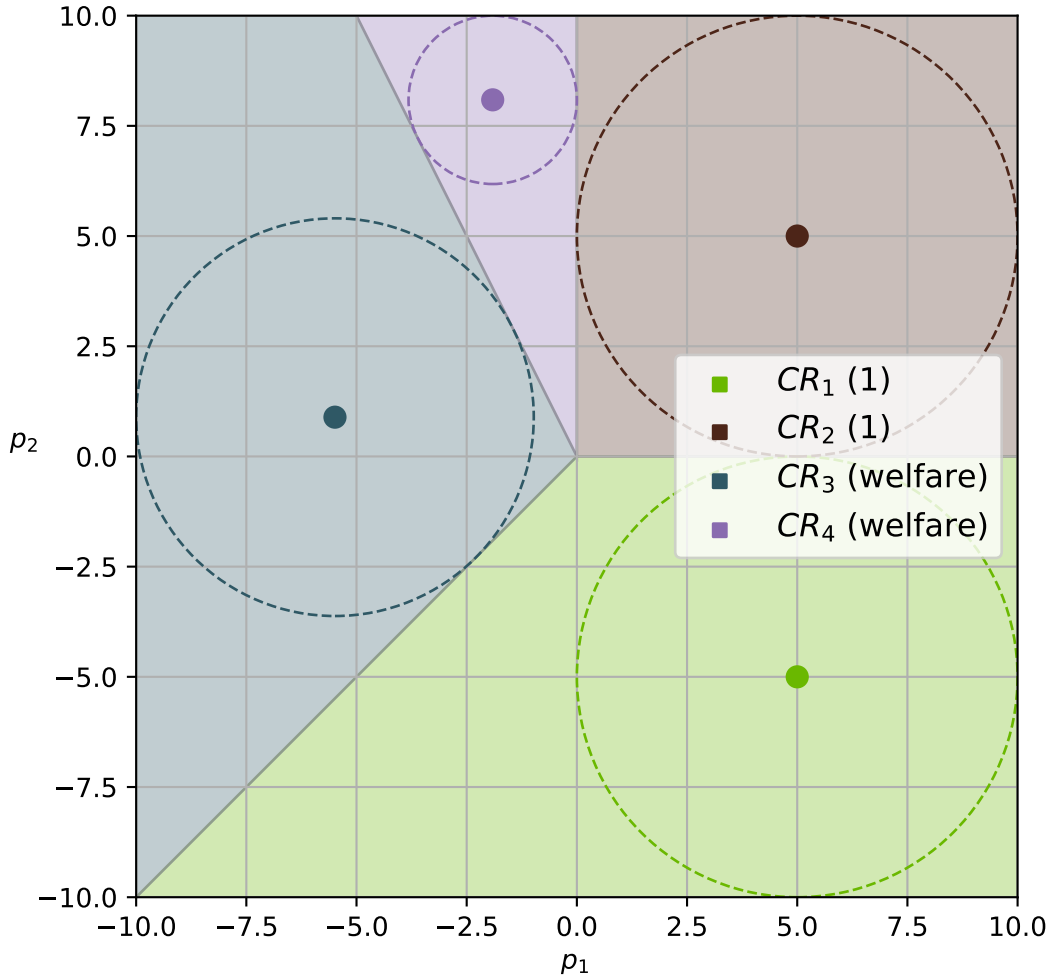
- **Critical regions**: use **min-norm GNE** solution criterion to split



(1=single solution, ∞ = infinitely-many solutions, min = min-norm GNE solution)

EXAMPLE: SIMPLE 2-AGENT GNEP

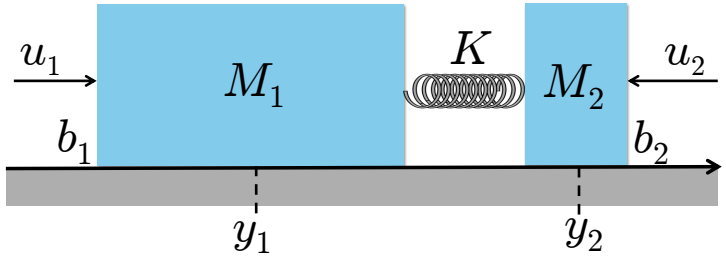
- **Critical regions**: use **welfare GNE** solution criterion to split



(1=single solution, ∞ = infinitely-many solutions, welfare = welfare GNE solution)

EXAMPLE: GAME-THEORETIC MPC

- **2-mass-spring-damper system** with $N = 2$ agents, each controlling one force (u_1 or u_2)



- $M_1 = 3, M_2 = 1, K = 0.5, \beta_1 = 1.5, \beta_2 = 1$ (MKS units)

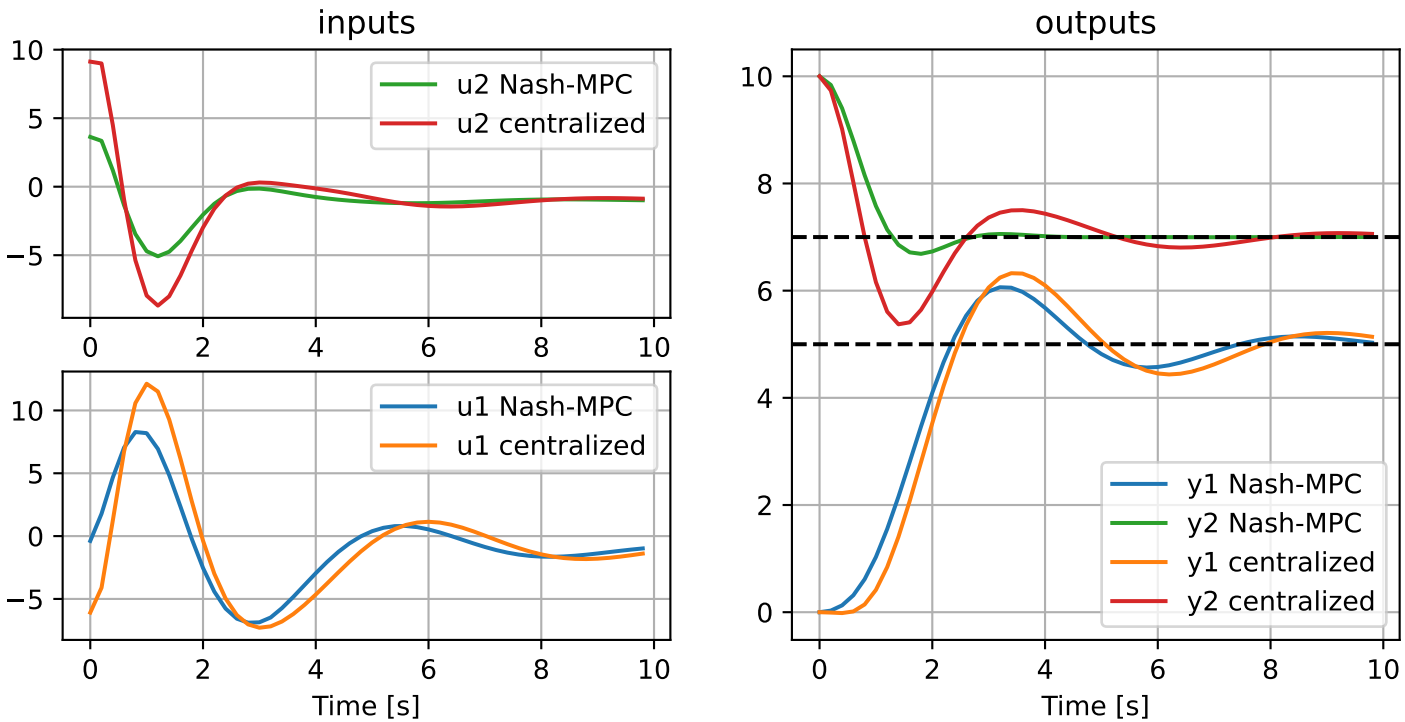
- Each agent minimizes its own MPC cost over a horizon of $N = 10$ steps:
  - **Agent 1**: $\min \sum (e_1 - e_2)^2 + (\Delta u_1 + \Delta u_2)^2$ (same tracking error, opposite Δu's)
  - **Agent 2**: $\min \sum e_2^2 + 0.1(\Delta u_2)^2$ (only on its own output and input increment)

- **State constraint**: $y_2 - y_1 \geq 1$ imposed over the first $N_c = 3$ steps

- **Parameters**: $p = (z(t), u(t-1), r(t)) \in \mathbb{R}^{4+2+2}$

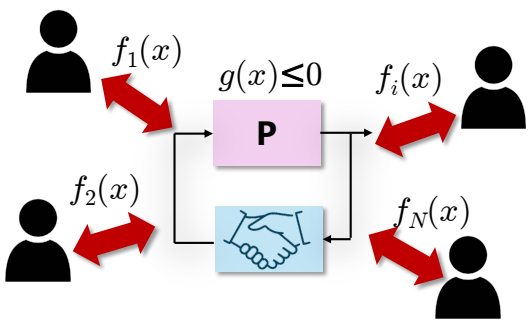
EXAMPLE: GAME-THEORETIC MPC

- Solution: 4 critical regions with unique GNE, 3 with infinitely-many GNEs
- Split the latter using **minimum-norm criterion** -> **19 critical regions**
- Compare with **centralized MPC** minimizing sum of costs + state constraints



- Other criteria (v-GNE, minimum-norm, welfare) give similar closed-loop results

NASHOPT LIBRARY



Contents based on the following paper:

A. Bemporad, "*NashOpt: A Python library for computing generalized Nash equilibria and game design*," *arXiv, 2025*

LIBRARY CONTENTS

- Solve **nonlinear GNEs** via nonlinear least-squares (=minimize KKT residuals)
- Find (one or more) GNEs of **linear–quadratic games** via MILP
- Solve **game-design** problems, such as **inverse-games**
- Solve **game-theoretic control** problems (**LQR** and **MPC**)

```
pip install nashopt
```

NashOpt

GENERAL PARAMETRIC GNEP

- General **parametric GNEP** with N agents:

$$\begin{aligned} \min_{p\in P} \quad & J(x^\star(p),p) & \\ \text{s.t.} \quad & x_i^\star(p) \in \arg\min_{x_i} & f_i(x,p) \\ & \text{s.t.} & g(x,p)\le 0,\ h(x,p)=0 \\ & & \ell_i \le x \le u_i \\ & & x_{-i} = x^\star_{-i}(p),\ i=1,\dots,N \end{aligned}$$

- Special cases:
  - **inverse game** design: $J(x^\star,p) = \|x^\star - x_{\rm des}\|_2^2$
  - **feasibility**: $J(x^\star,p)=0$, to find any p such that a GNE $x^\star(p)$ exists
  - **single GNEP**: fix $p=p_0$ (or no p), no cost J

OPTIMALITY CONDITIONS

- $x^\star(p)$ must satisfy the joint **Karush-Kuhn-Tucker (KKT)** (necessary) conditions

$$R(z,p) = \begin{bmatrix} \nabla_{x_i} f_i(x,p) + \nabla_{x_i} g(x,p)^\top \lambda_i + \nabla_{x_i} h(x,p)^\top \mu_i - v_i + y_i \\ \phi(\lambda_{i,j}, -g_j(x,p)) \\ \phi(v_{i,k}, x_{i,k} - \ell_{i,k}) \\ \phi(y_{i,k}, u_{i,k} - x_{i,k}) \\ h(x,p) \end{bmatrix} = 0$$

where $z = (x, \lambda, \mu, v, y)$ and $\phi(a,b) = 0 \quad \Leftrightarrow \quad a \geq 0,\ b \geq 0,\ ab = 0$
(e.g., **Fischer–Burmeister** function $\phi(a,b) = \sqrt{a^2+b^2} - a - b$)

- Solution methods:

single GNEP

$z^\star(p_0) \in \arg\min_z \frac{1}{2}\|R(z,p_0)\|_2^2$

(**nonlinear least-squares problem**)

game-design problem

$\min_{p\in P, z} J([I\ 0]z, p) + \frac{\rho}{2}\|R(z,p)\|_2^2$

(**nonlinear optimization problem**), $\rho \gg 1$

EXAMPLE: SINGLE GNEP

```
import jax
import jax.numpy as jnp
from nashopt import GNEP

sizes = [2, 1, 1]       # [n1, n2, n3]

def f1(x): # Agent 1 objective:
    return jnp.sum((x[0:2] - jnp.array([1.0, -0.5]))**2)

def f2(x): ...

def f3(x): ...

# Shared constraint g(x)<=0:
def g(x):
    return jnp.array([x[3] + x[0] + x[2] - 2.0])

lb=np.zeros(4) # lower bounds
ub=np.ones(4) # upper bounds

gnep = GNEP(sizes, f=[f1,f2,f3], g=g, ng=1, lb=lb, ub=ub, variational=False)
sol = gnep.solve()

print(sol.x) # x* = [ 1.   0.   0.    0.5]
```

EXAMPLE: GAME-DESIGN PROBLEM

- Solving a game-design problem:

```
from nashopt import ParametricGNEP

pgnep = ParametricGNEP(sizes, npar=2, f=f, g=g, ng=1, lb=lb, ub=ub,
                       Aeq=Aeq, beq=beq, Seq=Seq, h=h, nh=nh)

sol = pgnep.solve(J, pmin, pmax)
```

- Add ℓ_1/ℓ_2 regularization term $\alpha_1\|x\|_1 + \alpha_2\|x\|_2^2$ on J:

```
sol = pgnep.solve(J, pmin, pmax, alpha1=alpha1, alpha2=alpha2)
```

LINEAR QUADRATIC GNEPS

- **Linear-quadratic** game-design problem:

$$
\begin{array}{lll}
\min_{p,x^\star} & & J(x^\star,p) \\
\text{s.t.} & x_i^\star(p) \in \arg\min_{x_i} & f_i(x,p) = \frac{1}{2}x^\top Q^i x + (c^i + F^i p)^\top x \\
& \text{s.t.} & Ax \le b + Sp,\ A_{\rm eq}x = b_{\rm eq} + S_{\rm eq}p \\
& & \ell_i \le x \le u_i \\
& & x_{-i} = x^\star_{-i}(p),\ i = 1,\dots,N
\end{array}
$$

- $x^\star(p)$ can be characterized via **mixed-integer linear inequalities**, by introducing binary vars for modeling complementarity constraints (using the *big-M* method)

- If $J(x^\star,p)$ is convex quadratic $\rightarrow$ **MIQP problem**. If $J(x^\star,p)$ is piecewise affine and convex (or zero)$\rightarrow$ **MILP problem**

- **Multiple GNEs** can be extracted sequentially by imposing "no-good" constraints, that prevent combinations of active constraints already found

EXAMPLE: LINEAR QUADRATIC GAME

- Solving a single linear-quadratic GNEP:

```
from nashopt import GNEP_LQ

gnep = GNEP_LQ(sizes, Q, c, F, lb=lb, ub=ub, pmin=pmin, pmax=pmax,
               A=A, b=b, S=S, M=1e4, variational=variational, solver='highs')
sol = gnep.solve()
x = sol.x
```

Available solvers: `highs` (MILP), `gurobi` (MILP/MIQP), `prox_admm`[1]

- Get multiple GNEs (MILP only):

```
sol = gnep.solve(max_solutions=10)
```

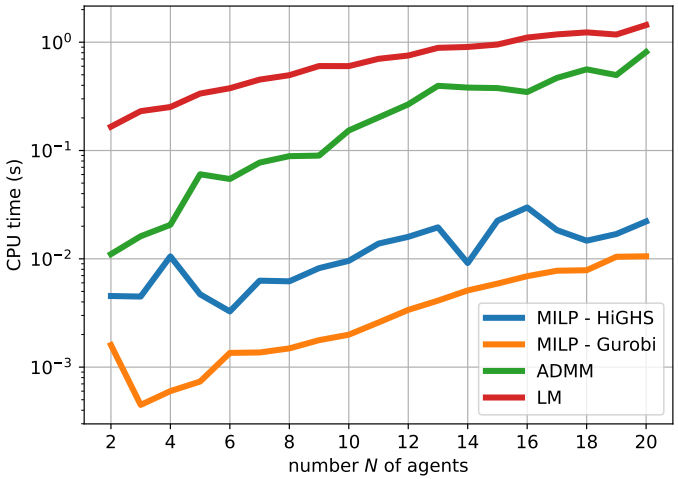
- Solving game-design problem $\min_p \frac{1}{2}p'Q_J p + c_J' p$ under $x^\star(p)$ being a GNE:

```
gnep_lq = GNEP_LQ(sizes, ... Q_J=Q_J, c_J=c_J, ...)
```

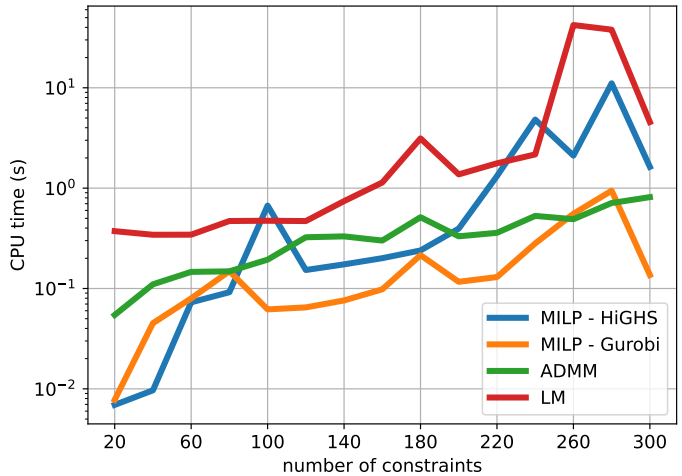
[1] Proximal ADMM method of (Börgens, Kanzow, 2021), only for single vGNEs

LQ GAMES - CPU TIME

- Compare solution methods on random LQ-GNEPs:



N agents, 2 variables each, $2N$ shared inequalities



$N = 5$ agents, 2 variables each, increasing shared inequalities

- Many agents, few constraints: prefer MILP (small number of binary variables)
- Few agents, many constraints: proximal ADMM could be preferable

NON-COOPERATIVE LQR

- **LQR** gain K_i for given K_{-i} is approximated by solving the algebraic Riccati equation (ARE) by fixed-point iterations:

$$\begin{aligned} K_i^k &= (R_i + B_i^T X_i^k B_i)^{-1} B_i^T X_i^k (A - B_{-i} K_{-i}) \\ A_k &= A - B_{-i} K_{-i} - B_i K_i^k \\ X_i^{k+1} &= Q_i + A_k^T X_i^k A_k + (K_i^k)^T R_i K_i^k \end{aligned}$$

  for $k = 1, \ldots, N_{\mathrm{LQR}}$, starting from $X_i^0 = Q_i$ (N_{LQR} large enough)

- Define game with agents' costs $f_i(K_i, K_{-i}) = \|K_i^{N_{\mathrm{LQR}}}(K_{-i}) - K_i\|_F^2$ and solve it via nonlinear least-squares method

- **Alternative method**: solve the coupled discrete-time algebraic Riccati equations iteratively (Nortman, Monti, Sassano, Mylvaganam, 2024)

EXAMPLE: NON-COOPERATIVE LQR

- Compute Nash LQR gains K_i via nonlinear least-squares method:

```
from nashopt import NashLQR

nash_lqr = NashLQR(sizes, A, B, Q, R, dare_iters=50)
sol = nash_lqr.solve(method = 'residual')
sol.K_Nash=K_Nash
```

- Compute instead K_i by solving directly the coupled AREs iteratively:

```
sol = nash_lqr.solve(method='riccati', riccati_iters=100, stop_tol=1e-5)
```

(usually this method is much faster)

GAME-THEORETIC MPC

- **Game-theoretic MPC** problem with input/output constraints:

$$
\begin{aligned}
(\Delta u_i, \epsilon_i) \in \arg\min \quad & \left(\sum_{k=0}^{T-1} (y_{k+1} - r(t))^\top Q_{y,i} (y_{k+1} - r(t)) + \Delta u_{i,k}^\top Q_{\Delta u,i} \Delta u_{i,k} \right) + q_{\epsilon,i} \epsilon_i \\
\text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \qquad\qquad y_{k+1} = Cx_{k+1} \\
& u_{k,i} = u_{k-1,i} + \Delta u_{k,i} \qquad\qquad u_{-1} = u(t-1) \\
& \Delta u_{\min} \le \Delta u_k \le \Delta u_{\max} \qquad\qquad u_{\min} \le u_k \le u_{\max} \\
& y_{\min} - \textstyle\sum_{i=1}^{N} \epsilon_i \le y_{k+1} \le y_{\max} + \sum_{i=1}^{N} \epsilon_i \qquad\qquad \epsilon_i \ge 0 \\
& i = 1, \ldots, N, \; k = 0, \ldots, T-1.
\end{aligned}
$$

- Agent #i's optimization variables: $(\Delta u_{i,0}, \ldots, \Delta u_{i,T-1}, \epsilon_i)$

- **LQ-GNE** problem solved at each time t via MILP, given the fixed parameter $p = (x(t), u(t-1), r(t))$

EXAMPLE: GAME-THEORETIC MPC

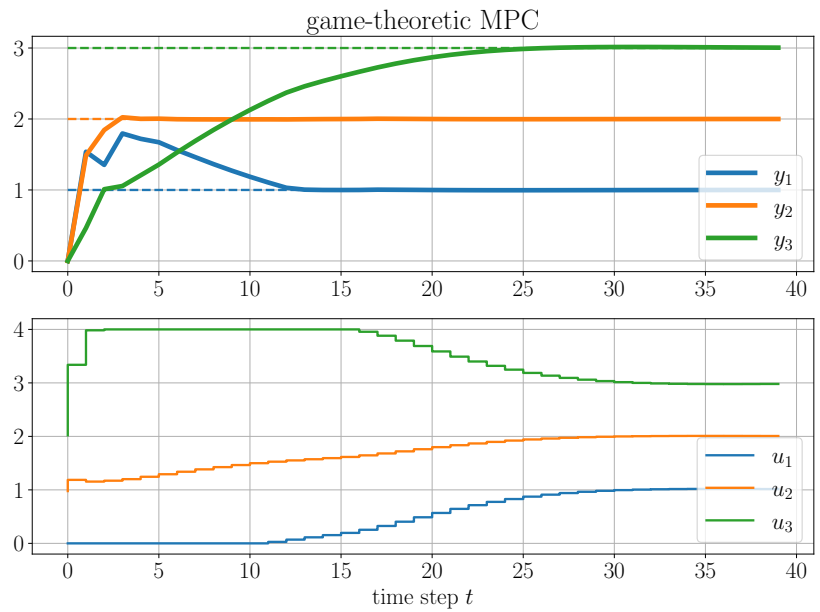
- Linear system: $x \in \mathbb{R}^6, u \in \mathbb{R}^3, y \in \mathbb{R}^3$. Agent i controls u_i and only weights y_i

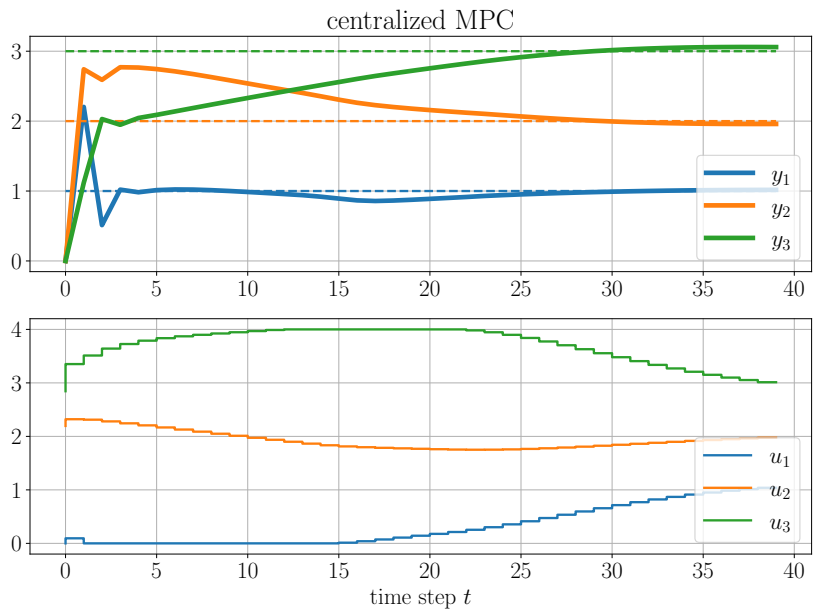
```
from nashopt import NashLinearMPC

nash_mpc = NashLinearMPC(sizes, A, B, C, Qy, Qdu, T, ymin=ymin, ymax=ymax,
                         umin=umin, umax=umax, dumin=dumin, dumax=dumax, Qeps=Qeps)
```

- At each time t, compute the control move $u(t)$:

```
sol = nash_mpc.solve(x, u1, r)
u = sol.u
```





CONCLUSIONS

CONCLUSIONS

- We have described different optimization methods to compute GNEs:
  - **Active-learning** method for general GNEPs from **best response data**
  - **Multiparametric programming** for **linear-quadratic** GNEPs
  - **Nonlinear LS/MIP** + other solution methods for **game-design** and **control** problems

- **Current research:**
  - New solution algorithms for (convex) GNEPs
  - Learn GNEs from data
  - Stabilizing game-theoretic MPC

(contact me if interested)

Funded by the European Union
European Research Council
Established by the European Commission
ERC Advanced Grant "COMPACT" (2024-2029)