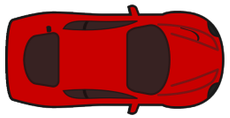
iLQGames.jl: Rapidly Designing and Solving Differential Games in Julia

Lasse Peters and Zachary N. Sunberg



General-Sum Differential Games

Joint Dynamics

$$\dot{x} = f(t, x, u_1, \ldots, u_N)$$

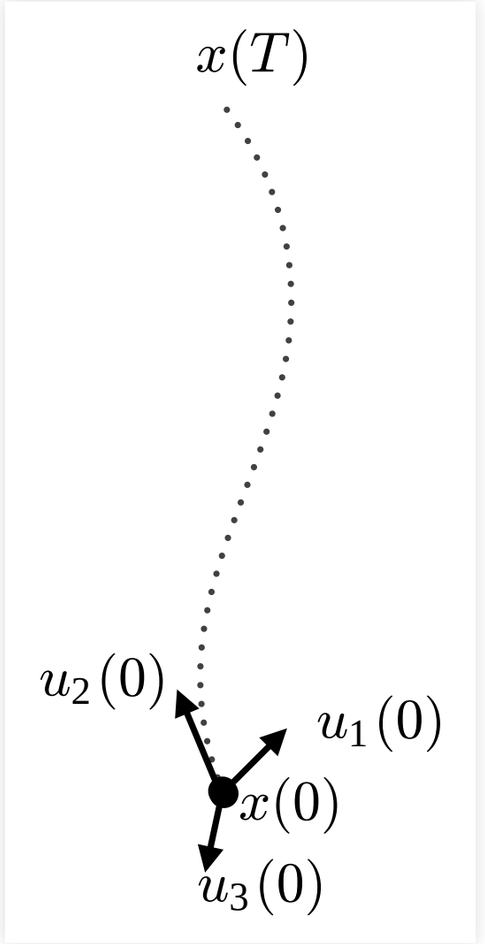
Cost for player i

$$J_i = \int_0^T g_i(t, x, u_1, \ldots, u_N)dt$$

Strategy of player i

$$u_i(t) = \gamma_i(t, x)$$

Solutions: Noncooperative Equilibria



iLQGames.jl

Challenges for application of differential games

- computational complexity
- efficient implementation of existing algorithms
- conceptual complexity of interfaces to describe the problem

iLQGames.jl

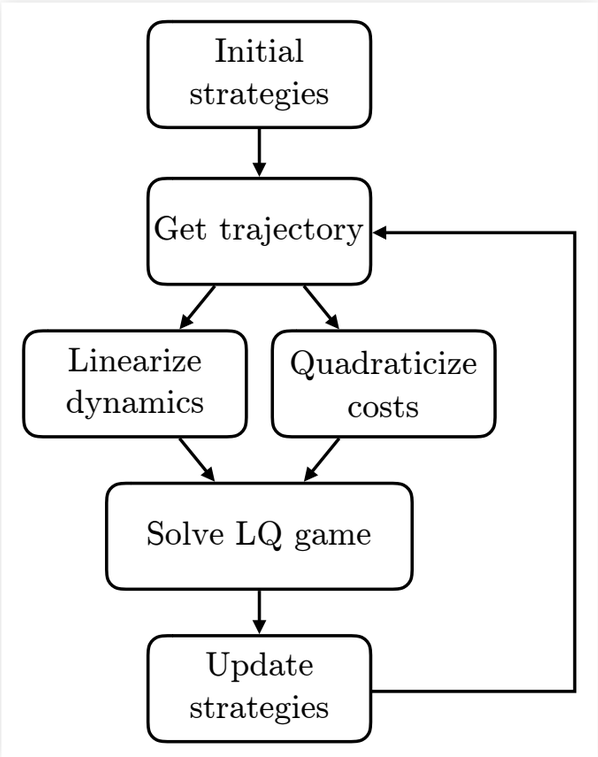
- lightweight interface for describing differential games
- efficient implementation of a game-solver[1]
- tools for visualization of solutions

[1] game solver

Fridovich-Keil, D., Ratner, E., Peters L., Dragan, A.D., & Tomlin, C.J. Tomlin (2019). “Efficient Iterative Linear-Quadratic Approximations for Nonlinear Multi-Player General-Sum Differential Games.” ArXiv abs/1909.04694

Julia Language Features Enabling Flexibility and Performance

Iterative LQ Games



LQ approximation

$$\frac{d}{dt}\delta x \approx A(t)\delta x + \sum_{i=1}^{N} B_i(t)\delta u_i$$

$$\delta g_i \approx \frac{1}{2}\delta x^T Q_i(t)\delta x + \delta x^T l_i(t) + \frac{1}{2}\sum_{j=1}^{N} \delta u_j^T R_{ij}(t)\delta u_j + \delta u_j^T r_{ij}(t)$$

Helpful language features

- LQ approximations computed via auto-differentiation[2]
- dynamic dispatch for specialization of subroutines (e.g. [3])
- compiled and optimized code: e.g., stack-allocated inner loop

[2] auto-differentiation

Revels, J., Lubin, M., & Papamarkou, T. (2016). "Forward-Mode Automatic Differentiation in Julia." ArXiv abs/1607.07892

[3] solver for games with feedback-linearizable dynamics

Fridovich-Keil, D., Royo, V.R., & Tomlin, C.J. (2019). "An Iterative Quadratic Method for General-Sum Differential Games with Feedback Linearizable Dynamics." ArXiv abs/1910.00681

Benchmark Result

	(a) LQ (2P, 2D)	(b) NL (3P, 12D)	(c) FBL (3P, 12D)
LQ-Python	20.800 ms	n/a	n/a
iLQGames-C++	0.3490 ms	16.27 ms	13.25 ms
iLQGames.jl-MD	0.0044 ms	7.19 ms	3.98 ms
iLQGames.jl-AD	n/a	63.57 ms	52.50 ms

iLQGames-C++ and iLQGames.jl-MD use manually specified LQ-approximations.
iLQGames-AD uses auto-differentiation to for lq-approximations.

$\Rightarrow$ achieves execution times of a comparable C++ implementation

Demo

Demo: Design and Solution of a 2-Player Game

Dynamics: 4D Unicycle

$$\dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ a \end{bmatrix}$$

$$u_1 = \omega \qquad u_2 = a$$

Costs: P1 stay at origin, P2 keep v at 1m/s

$$g_1(t, x, u_1, u_2) = p_x^2 + p_y^2 + u_1^2$$

$$g_2(t, x, u_1, u_2) = (v-1)^2 + u_2^2$$

Setting up a problem in iLQGames.jl

```
# constants: number of {states,inputs}, sampling time, horizon
nx, nu, ΔT, game_horizon = 4, 2, 0.1, 200

# setup a dynamical system
struct Unicycle <: ControlSystem{ΔT,nx,nu} end
dx(cs::Unicycle, x, u, t) = SVector(x[4]cos(x[3]),
                                    x[4]sin(x[3]),
                                    u[1],
                                    u[2])
dynamics = Unicycle()
```

```
# player-1 wants the unicycle to stay close to the origin,
# player-2 wants to keep the speed of the unicycle close to 1 m/s
costs = (FunctionPlayerCost((g,x,u,t) -> (x[1]^2+x[2]^2+u[1]^2)),
         FunctionPlayerCost((g,x,u,t) -> ((x[4]-1)^2+u[2]^2)))

# indices of inputs that each player controls
player_inputs = (SVector(1), SVector(2))
# the horizon of the game
g = GeneralGame(game_horizon, player_inputs, dynamics, costs)
```

Demo: Design and Solution of a 2-Player Game

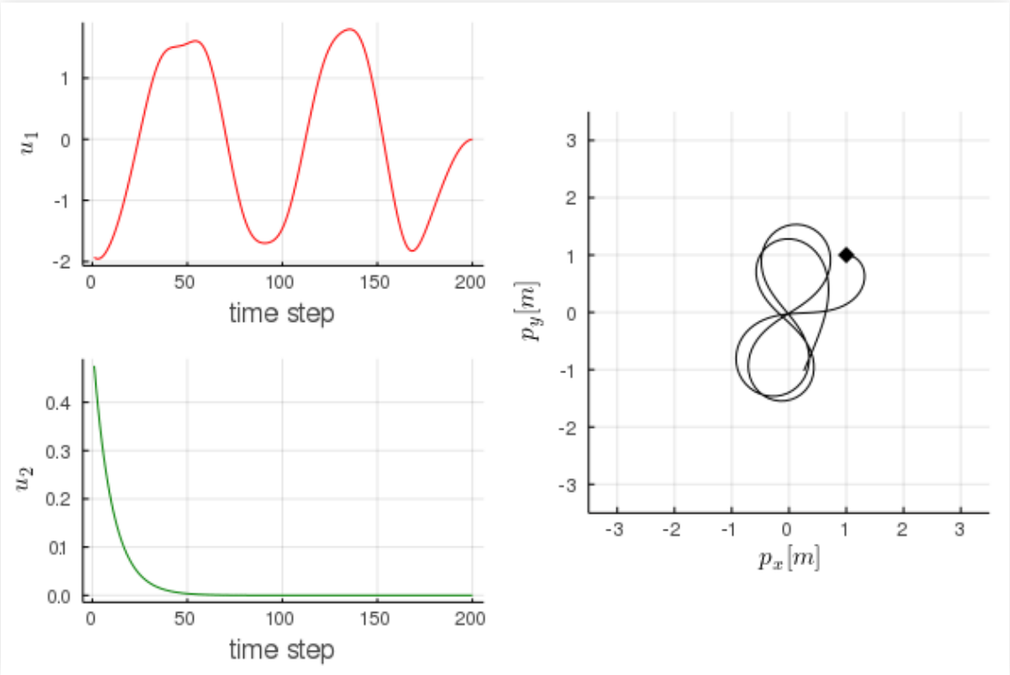
Solving the Game

```
solver = iLQSolver(g)
x0 = SVector(1, 1, 0, 0.5) # initial state
converged, trajectory, strategies = solve(g, solver, x0)
```

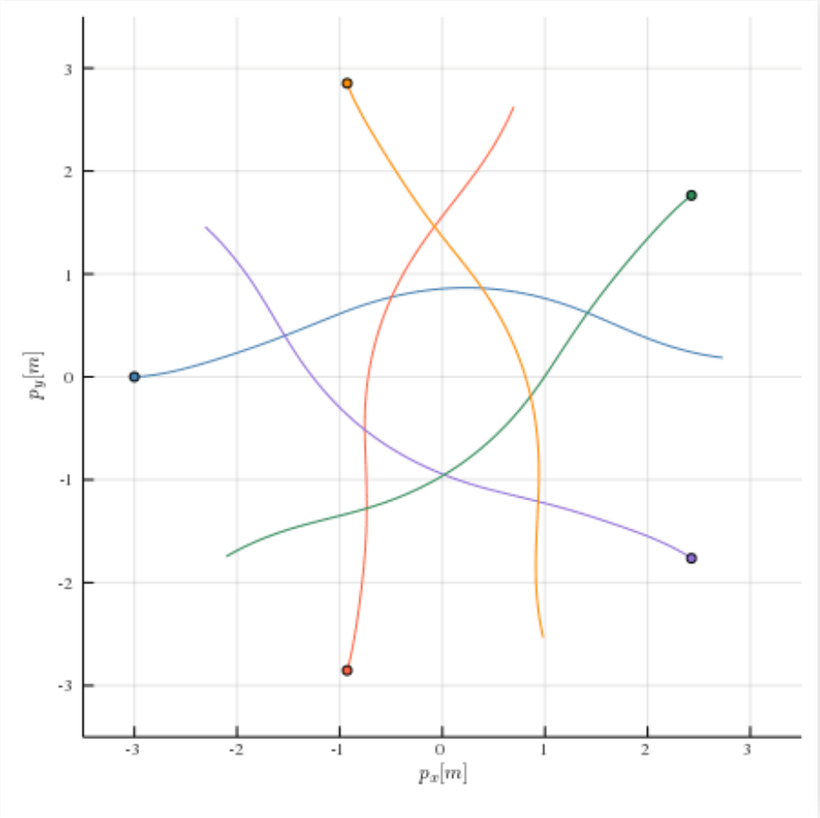
Visualizing the Solution

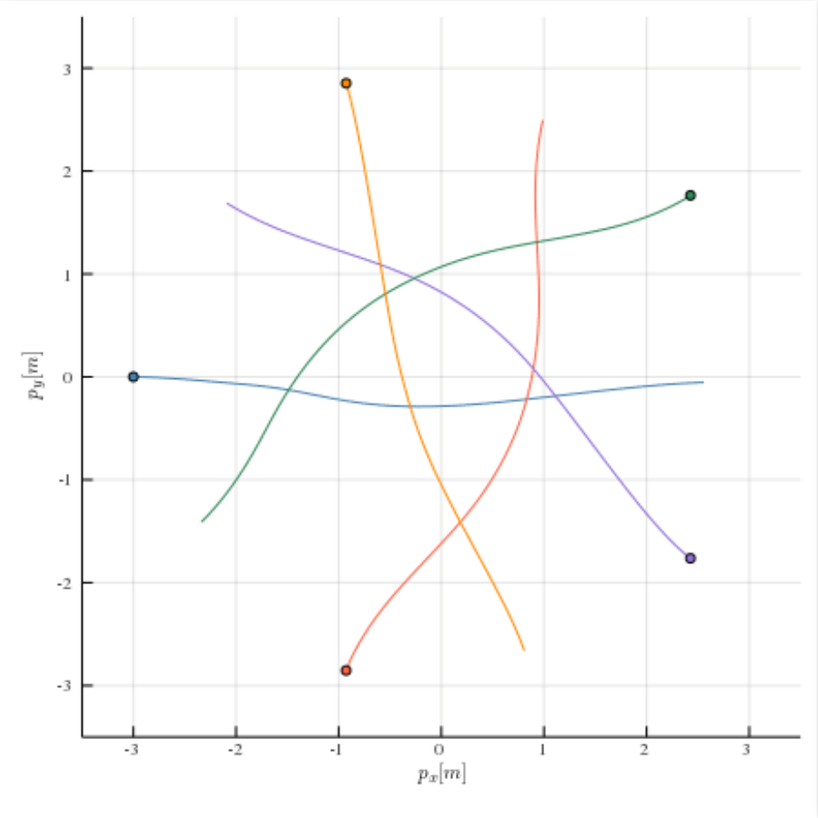
```
position_indices = tuple(SVector(1,2))
@animated(plot_traj(trajectory, g, [:red, :green], player_inputs),
          1:game_horizon, "minimal_example.gif")
```

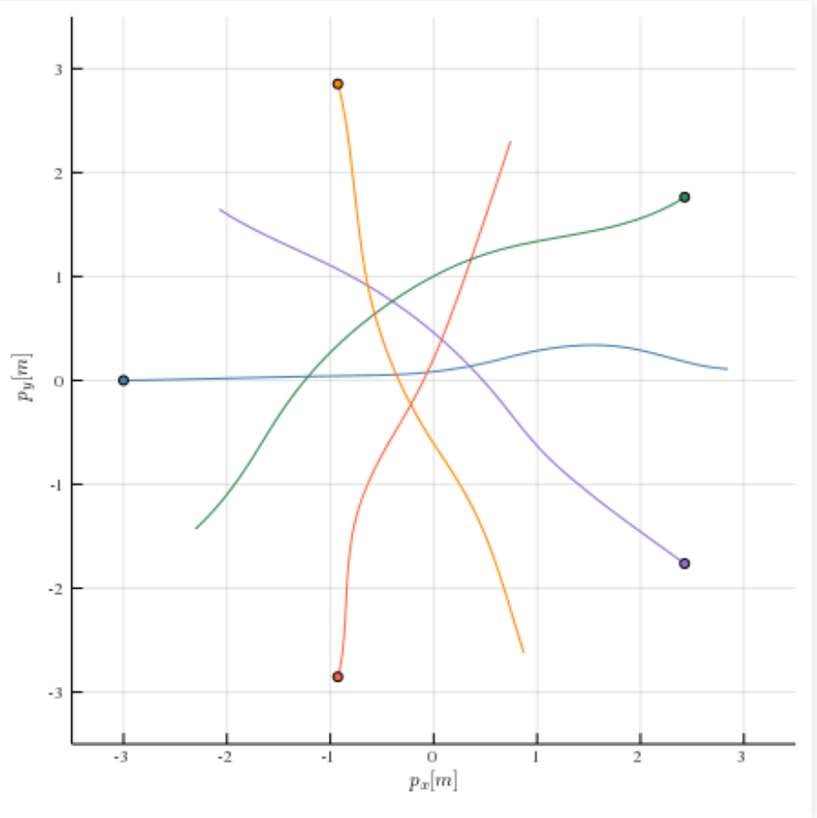
Game Solution



Examples of 5-Player Game Solutions







Thank You

Paper

Peters, L. & Sunberg Z. (2020). “iLQGames.jl: Rapidly Designing and Solving Differential Games in Julia.” ArXiv abs/2002.10185

Code

iLQGames.jl: github.com/lassepe/iLQGames.jl