Biblioteca para Cálculo de Conjuntos Invariantes com *Julia*

***Nicholas W. Lopes*[1]**
***Felipe R. Silva*[2]**
***Carlos E. T. Dórea*[3]**

[1] *Departamento de Engenharia de Computação, UFRN, <wallace.lopes.162@ufrn.edu.br>*

[2] *Departamento de Engenharia de Computação, UFRN, <felipe.rocha.702@ufrn.edu.br>*

[3] *Departamento de Engenharia de Computação, UFRN, <cetdorea@dca.ufrn.br>*

Resumo

Este artigo apresenta o desenvolvimento de uma biblioteca em *Julia* para o cálculo e a análise de conjuntos invariantes de sistemas dinâmicos lineares discretos. A ferramenta permite definir sistemas a partir de modelos em espaço de estados, simular trajetórias, verificar a invariância de conjuntos poliédricos considerando restrições e buscar conjuntos que satisfaçam as propriedades de invariância e contração. A implementação utiliza o Lema de Farkas para formular condições matemáticas e integra as bibliotecas JuMP, Polyhedra e CDDlib para a modelagem de problemas de otimização e a manipulação de poliedros. A proposta busca centralizar, em um único ambiente computacional, etapas anteriormente realizadas com diferentes ferramentas. Por meio de exemplos, são apresentadas as funcionalidades implementadas e sua aplicação a problemas de invariância, incluindo sistemas com atraso na entrada de controle.

Abstract

This paper presents the development of a *Julia* library for the computation and analysis of invariant sets for discrete-time linear dynamical systems. The library allows users to define systems using state-space models, simulate trajectories, verify the invariance of polyhedral sets under constraints, and search for sets satisfying invariance and contraction properties. The implementation uses Farkas' lemma to formulate mathematical conditions and integrates the JuMP, Polyhedra, and CDDlib libraries for optimization modeling and polyhedral manipulation. The proposed approach aims to consolidate, within a single computational environment, tasks that were previously performed using different tools. Examples illustrate the implemented functionalities and their application to invariant-set problems, including systems with input delays.

1 Introdução

O estudo da estabilidade de sistemas dinâmicos é fundamental para a teoria de controle, uma vez que permite analisar o comportamento das trajetórias do sistema ao longo do tempo e estabelecer condições para sua operação segura e previsível. Nesse contexto, a abordagem de conjuntos invariantes permite caracterizar regiões do espaço de estados nas quais as trajetórias permanecem confinadas sob a dinâmica do sistema. Essas regiões são definidas por hiperplanos de conjuntos convexos, em particular, esse trabalho enfatiza os conjuntos hiperpoliédricos (*hyperpolyhedral sets*), que despertam interesse devido

à sua capacidade de representar restrições sobre as variáveis de estado e de entrada. O cálculo dessas regiões, entretanto, pode exigir procedimentos computacionais específicos, motivando o desenvolvimento de ferramentas para sua determinação.

Na literatura, diferentes formulações de problemas de otimização têm sido propostas para a determinação de conjuntos poliédricos invariantes. Em trabalhos anteriores (DóREA; OLARU; NICULESCU, 2023), a modelagem dos problemas era realizada por meio da linguagem AMPL (*A Mathematical Programming Language*) (FOURER; GAY; KERNIGHAN, 1990), e sua solução era obtida com o auxílio de solvers de otimização como o Knitro (BYRD; NOCEDAL; WALTZ, 2006). Posteriormente, os resultados eram processados no MATLAB (The MathWorks, Inc.,), utilizando ferramentas específicas para manipulação de poliedros. Embora essa abordagem permita combinar recursos especializados, ela distribui as etapas do procedimento entre diferentes ambientes computacionais, podendo aumentar a complexidade de implementação e manutenção. Nesse contexto, a utilização de um único ambiente de programação surge como uma alternativa para integrar a modelagem, a resolução dos problemas de otimização e as simulações com os poliedros resultantes.

Com esse objetivo, este trabalho utiliza *Julia* (BEZANSON et al., 2017), uma linguagem de programação de código aberto que apresenta recursos particularmente adequados à computação científica e à otimização matemática. Seu ecossistema dispõe de bibliotecas que atendem às diferentes etapas do cálculo de conjuntos invariantes. Entre elas, destaca-se JuMP (LUBIN et al., 2023), utilizada para a modelagem de problemas de otimização, e Polyhedra (LEGAT, 2023), que fornece uma interface para a representação e a manipulação de poliedros. A integração dessas bibliotecas permite explorar recursos já disponíveis no ecossistema *Julia* e reduzir a necessidade de implementar, do zero, funcionalidades matemáticas consolidadas.

Diante disso, este trabalho apresenta o desenvolvimento de um pacote (*package*) em *Julia* para auxiliar o cálculo de conjuntos invariantes, integrando bibliotecas existentes a funções desenvolvidas especificamente para o problema considerado. A implementação busca centralizar as etapas computacionais envolvidas no procedimento e oferecer uma estrutura modular, de código aberto, que facilite a manutenção, a extensão das funcionalidades e a contribuição de outros desenvolvedores. Dessa forma, pretende-se disponibilizar uma ferramenta que apoie estudos sobre conjuntos invariantes e favoreça o desenvolvimento de novas funcionalidades nesse contexto.

2 Metodologia

2.1 Fundamentação Teórica

Inicialmente, definimos o escopo e a estrutura matemática adotados neste trabalho. Consideram-se sistemas dinâmicos discretos no tempo, descritos na representação em espaço de estados por:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

em que $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}$ e $y(k) \in \mathbb{R}$ representam, respectivamente, o vetor de estados, a entrada de controle e a saída do sistema. As matrizes A, B e C descrevem a dinâmica dos estados, a influência da entrada de controle e a relação entre os estados e a saída.

Um conjunto $S \subseteq \mathbb{R}^n$ é dito positivamente invariante em relação ao sistema definido em 1 se, para qualquer condição inicial $x(0) \in S$, a trajetória do sistema permanecer em S para todo instante $k \geq 0$, isto é,

$$x(0) \in S \Rightarrow x(k) \in S, \quad \forall k \geq 0.$$

Essa propriedade garante que, uma vez que o estado do sistema pertença a S, ele permanecerá nessa região ao longo de toda a evolução temporal. Uma propriedade mais restritiva pode ser estabelecida por meio da definição de conjuntos λ-contrativos positivamente invariantes, para os quais existe um fator de contração $0 < \lambda < 1$ tal que

$$x(k) \in S \Rightarrow x(k+1) \in \lambda S. \tag{2}$$

Essa condição garante que o $x(k)$ converge para a origem quando $k \to \infty$, desde que S seja limitado e fechado. Assim, além de assegurar a permanência da trajetória em uma região, a propriedade de contração permite estabelecer resultados de convergência assintótica.

No contexto de problemas de controle com restrições, é comum que os limites impostos às variáveis de estado e de entrada sejam expressos por meio de desigualdades lineares. Essas restrições podem ser representadas geometricamente por conjuntos poliédricos convexos. Em particular, um conjunto de restrições no espaço de estados pode ser descrito na forma

$$\Omega = \{x \in \mathbb{R}^n : Fx \leq w\}, \quad F \in \mathbb{R}^{f \times n}, \quad w \in \mathbb{R}^f.$$

A representação poliédrica é especialmente interessante porque permite transformar determinadas propriedades geométricas em condições algébricas verificáveis. Em particular, o Lema de Farkas Estendido fornece ferramentas para estabelecer condições de inclusão entre poliedros. Essa possibilidade é fundamental para o estudo de conjuntos invariantes, pois permite formular condições suficientes para garantir que a evolução do sistema permaneça dentro das restrições impostas.

Dessa forma, a combinação da representação poliédrica com ferramentas de verificação algébrica fornece uma base sistemática para o estudo de sistemas de controle sujeitos a restrições. Mais a diante abordaremos como essas definições são utilizadas em projetos de controladores baseados em problemas de otimização com restrições.

Para verificar inclusão de poliedros, utilizamos o Lema de Farkas Estendido. a implicação $F_2x \leq w_2 \Rightarrow F_1x \leq w_1$ é válida se existir uma matriz D não negativa tal que:

$$\begin{aligned} DF_2 &= DF_1, \\ Dw_2 &\leq Dw_1. \end{aligned}$$

Ou seja, para $\Omega_1 = \{x : F_1x \leq w_1\}$ e $\Omega_2 = \{x : F_2x \leq w_2\}$, a existência de D garante $\Omega_2 \subset \Omega_1$. Isso é especialmente útil quando há restrições sobre os estados, pois, garantindo que o poliedro invariante esteja contido nelas, o sistema nunca violará as restrições para um estado dentro desse poliedro.

Mais adiante, o Lema de Farkas também é utilizado para garantir a invariância de poliedros, como já mencionado anteriormente. Para um sistema com dinâmica $x(k+1) = Ax(k)$, por exemplo, essa propriedade pode ser escrita como $Fx(k) \leq w \Rightarrow FAx(k) \leq w$. Essa expressão se assemelha bastante à inclusão de poliedros descrita acima, de forma que, para que o poliedro convexo Ω seja λ-contrativo e positivamente invariante, deve existir uma matriz H não negativa tal que:

$$\begin{aligned} HF &= FA, \\ Hw &\leq \lambda w. \end{aligned}$$

Essas condições são necessárias e suficientes para garantir a propriedade de invariância e podem ser adaptadas para outras dinâmicas de $x(k+1)$.

2.2 Design da Biblioteca

O paradigma de despacho múltiplo (*multiple dispatch*) é uma das principais características de *Julia*. Nesse paradigma, a seleção do método a ser executado depende dos tipos de todos os argumentos da chamada da função. Essa abordagem difere do mecanismo tradicional de despacho simples, comum em linguagens orientadas a objetos, no qual a seleção do método depende principalmente do tipo do objeto receptor. Dessa forma, *Julia* permite implementar padrões de programação orientada a objetos, além de oferecer outras possibilidades de organização e extensão do código. Mais informações sobre os benefícios do *multiple dispatch* podem ser encontradas na discussão apresentada em (The Julia Programming Language, 2020).

Diferentemente das linguagens orientadas a objetos tradicionais, *Julia* organiza seus dados por meio de tipos, incluindo tipos abstratos e concretos, que podem estabelecer relações hierárquicas de subtipagem. Nesse contexto, utilizamos estruturas definidas com a palavra-chave `struct` para estabelecer os tipos concretos que compõem a biblioteca, conforme representado na Figura 1. Essa organização busca tornar a arquitetura clara, modular e escalável, facilitando a manutenção do código e a incorporação de futuras contribuições.

O principal tipo definido é `InvariantPolyhedron`, que representa o objeto de estudo da biblioteca: um conjunto poliédrico λ-contrativo positivamente invariante, conforme definido na equação (2). A ideia é permitir que as funções relacionadas a esses conjuntos recebam objetos desse tipo. Para isso, seus campos incluem as matrizes F e w, que caracterizam o poliedro, o escalar λ, que representa o fator de contração, e o campo `system`, que identifica o sistema em relação ao qual o conjunto é invariante.

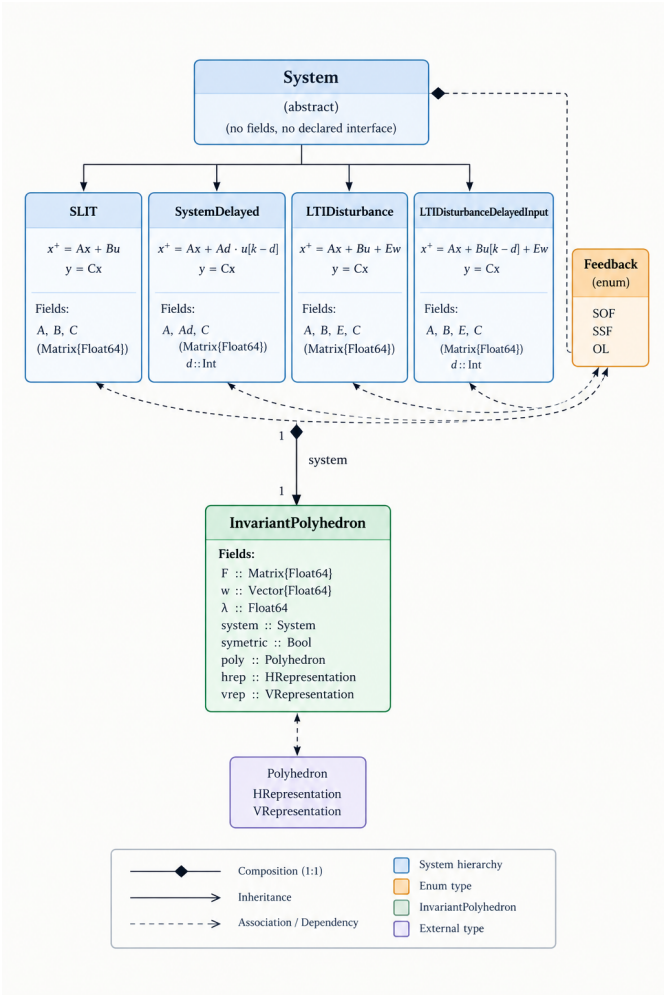


Figura 1: *Diagrama UML da estrutura de tipos da biblioteca*

Além desses campos, a estrutura armazena representações do poliedro fornecidas por uma biblioteca específica para computação poliédrica, que utiliza algoritmos da CDDlib (FUKUDA, 2022). Essa biblioteca permite trabalhar com representações por desigualdades lineares e por vértices e raios. Como cada representação pode ser mais adequada a determinadas operações, armazenamos ambas para evitar conversões recorrentes durante os cálculos.

Para representar os sistemas dinâmicos, definimos o tipo abstrato `System`, do qual derivam os tipos concretos que representam os sistemas suportados pela biblioteca. Além das matrizes da representação em espaço de estados, cada sistema possui uma matriz de ganhos G e um campo `feedback`, definido por um tipo enumerado (`@enum`), cujos valores correspondem às leis de controle descritas na Tabela 1.

Em geral, os ganhos do controlador não são conhecidos no momento da definição do sistema. Por esse motivo, disponibilizamos um construtor que recebe apenas as matrizes do modelo e inicializa o ganho com um valor unitário. Esse valor pode ser posteriormente atualizado por algoritmos responsáveis pelo cálculo dos ganhos do controlador.

Tipo	Lei de controle	Dinâmica em malha fechada
SOF	$u(k) = Gy(k) = GCx(k)$	$x(k+1) = (A + BGC)\,x(k)$
SSF	$u(k) = Gx(k)$	$x(k+1) = (A + BG)\,x(k)$

Tabela 1: *Tipos de realimentação e suas dinâmicas em malha fechada*

Para encontrar os ganhos do controlado um problema de otimização foi formulado por (BRIÃO et al., 2021), com as restrições dadas pelas condições do Lema de Farkas e como função objetivo o fator de contração λ a ser minimizado. Esse problema foi traduzido para o JuMP (LUBIN et al., 2023) por meio de uma função para cada tipo de sistema, adaptando as condições de invariância às dinâmicas específicas de cada um. Com isso, obteve-se uma função que recebe como entrada um conjunto poliédrico, um sistema de interesse e as restrições e que retorna, caso o conjunto seja invariante, o fator de contração associado. Caso não seja invariante, a função indica isso ao usuário.

Inicialmente, foi implementado um sistema linear invariante no tempo, descrito na equação (1). A partir desse modelo, foram consideradas extensões que contemplam um atraso na entrada de controle e a presença de uma perturbação. A mesma estrutura pode ser estendida para incorporar outros modelos dinâmicos, permitindo ampliar as funcionalidades da biblioteca de forma modular.

Vale ressaltar que os sistemas implementados até então foram sendo adicionados conforme necessário, pois essa biblioteca estava sendo desenvolvida para atacar o problema do segmento de referência para sistemas com atraso. Mais adiante abordaremos sobre funções específicas para esse propósito.

2.3 Verificação de Invariância Positiva

2.4 Determinação de Poliedros

E se não houver, de antemão, um poliedro candidato para testar a sua invariância? Nesse caso, pode-se utilizar o problema de otimização (BRIÃO et al.,

2021), mas agora considerando as matrizes que caracterizam o poliedro como variáveis do problema. Isso implica que o problema de otimização que antes era convexo passa a ser não convexo, tornando mais difícil a busca por um ótimo global.

Outra implicação é que passa a fazer sentido incorporar à função objetivo algum termo relativo ao tamanho do poliedro, pois, como mostrado por (DóREA; OLARU, 2022), há um compromisso entre o tamanho do poliedro e o fator de contração. Para tentar contornar isso, adiciona-se um fator de ponderação π à função objetivo, de forma que $obj = \pi\lambda - (1-\pi)\,\text{enl}(\cdot)$, em que $\text{enl}(\cdot)$ é uma função para maximizar o tamanho do poliedro.

Na primeira versão da biblioteca, foram implementadas duas funções $\text{enl}(\cdot)$. Na primeira, definem-se n vetores igualmente espaçados, cada um associado a um escalar γ_k, de forma a maximizar a soma desses escalares, como descreve a equação abaixo:

$$enl(.) = \frac{\sum_{k=1}^{n} \gamma_k}{n} \tag{3}$$

Além disso foi preciso adicionar mais n restrições ao problema para garantir que todos os vetores estejam contidos em Ω.

$$L * \gamma_k \mathbf{v}_k \leq \phi, k = 1, 2, ..., n \tag{4}$$

Para sistemas com apenas dois estados, a interpretação geométrica da estratégia pode ser ilustrada pela Figura 2.

Para a outra implementação da função $\text{enl}(\cdot)$, criamos um poliedro com o mesmo formato das restrições de estado e, com auxílio do *Lemma de Farkas*, adicionamos uma restrição no problema de otimização para que ele estivesse contido em Ω que queremos encontrar com a seguinte expressão.

$$\begin{aligned} Q\mathcal{S} &= F, \\ Q\mathbf{1} &\leq \gamma\mathbf{1} \end{aligned}$$

Dessa forma, para $\text{enl} = \gamma$, quando minimizamos γ estamos expandindo o poliedro no interior de Ω e consequentemente expandindo Ω também. Como pode ser visto na figura (3).

figure poliedros contidos

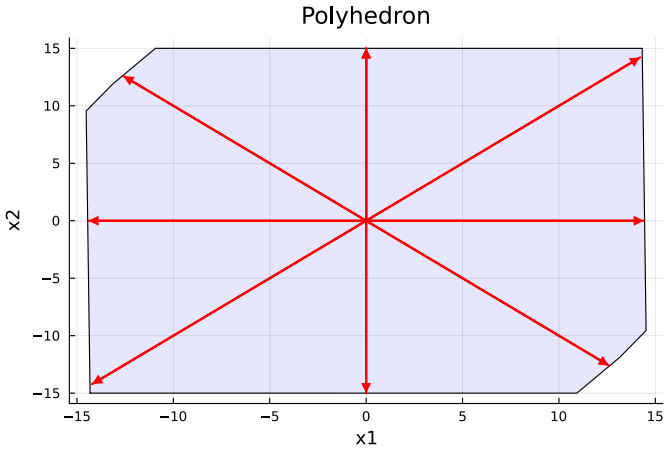


Figura 2: *Poliedro com os vetores utilizados para maximizar o seu tamanho*

2.5 Programas específicos para atraso

Considere o sistema a seguir:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k-d), \\ y(k) &= Cx(k) \end{aligned} \tag{5}$$

A entrada de controle u está atrasada em d amostras. O foco do trabalho principal que motivou a criação dessa biblioteca é sobre invariância com atraso, dessa forma foram desenvolvidos alguns programas auxiliares que lidam com problemas que surgem quando consideramos o atraso.

Utilizamos o conceito de $\mathcal{D}$-invariância *delay-dependent* (DóREA; OLARU; NICULESCU, 2023) que é uma abordagem menos conservadora que leva em conta o tamanho do atraso para calcular o conjunto invariante. Por outro lado, nessa abordagem, a invariância só é válida para um conjunto de condições iniciais, que pode ser calculado a partir do poliedro invariante e das matrizes do sistema original. Para tanto nós trabalhamos com o espaço expandido em que cada estado é uma amostra do sistema original. A ideia é encontrar vetores de condições iniciais $\mathbf{x}^T = [x(0), x(-1), ..., x(-d+1), x(-d)]$ para os quais

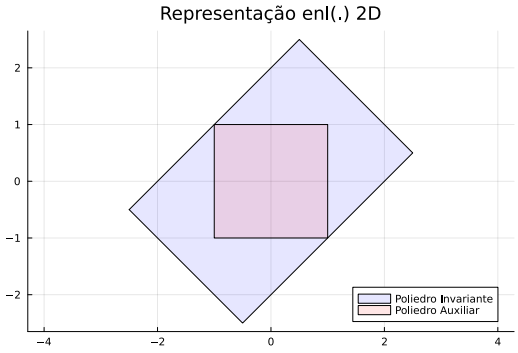


Figura 3: *Poliedro auxiliar contido no poliedro invariante*

a invariância é valida para o sistema original.

$$X(k+1) = \mathcal{A}X(k) = \begin{bmatrix} A & 0 & \cdots & 0 & A_d \\ I & 0 & \cdots & 0 & 0 \\ 0 & I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ x(k-1) \\ \vdots \\ x(k-d+1) \\ x(k-d) \end{bmatrix}$$

Pode ser demonstrado que para a condição inicial ser válida ela deve fazer parte do poliedro Ω_{adm} definido por

$$\begin{bmatrix} \mathcal{F} \\ \mathcal{F}\mathcal{A} \\ \mathcal{F}\mathcal{A}^2 \\ \vdots \\ \mathcal{F}\mathcal{A}^d \end{bmatrix} X(k) \leq \begin{bmatrix} \mathcal{W} \\ \mathcal{W} \\ \mathcal{W} \\ \vdots \\ \mathcal{W} \end{bmatrix}, \quad \mathcal{F} = \begin{bmatrix} F & 0 & \cdots & 0 & 0 \\ 0 & F & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & F & 0 \\ 0 & 0 & \cdots & 0 & F \end{bmatrix}.$$

O que o programa desenvolvido fez então foi montar essas matrizes estendidas, com base nas matrizes do sistema e no poliedro. Note que, quanto maior o atraso, maior a dimensão dessas matrizes, o que acaba tornando a computação mais custosa. Algo que fizemos para mitigar esses efeitos foi ir eliminando as redundâncias dessas matrizes no momento da criação delas. Dessa forma, conseguimos definir o conjunto de condições iniciais admissíveis de forma mais eficiente.

Uma vez que temos Ω_{adm} podemos partir para as simulações. Se o usuário já tiver uma condição inicial em mente ele pode testar se essas condições estão contidas em Ω_{adm} pela verificação da desigualdade. Por outro lado, se não houver nenhuma condição candidata, podemos escolher um dos vértices do poliedro de condições iniciais admissíveis, plotados na figura 4, como ponto de partida. Essa segunda abordagem é interessante pois não precisamos de um chute inicial para encontrar uma condição e em geral conseguimos pontos interessantes para simular, que se encontram perto dos limites do poliedro invariante encontrado.

.

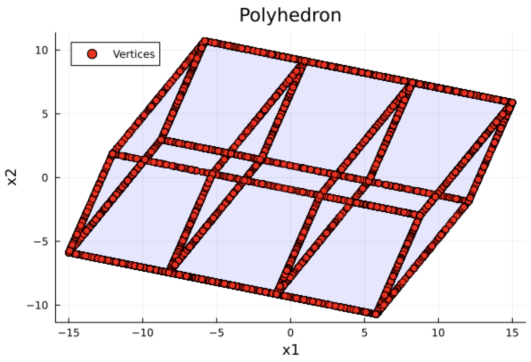


Figura 4: *x(0) de todos os vértices do poliedro de condições iniciais admissíveis, com o poliedro invariante de fundo*

3 Exemplos

Para mostrar um pouco mais sobre o fluxo de trabalho com essa biblioteca e como todo o processo que antes era dividido entre vários ambientes foi todo sintetizado em um só lugar, separamos um exemplo numérico. Mais demonstrações do uso juntamente com a documentação podem ser vistas em https://github.com/Nicholas-Wallace/Polyhedron .

Com o *Julia* instalado na máquina o usuário tem a opção de usar o REPL nativo da linguagem dentro do próprio terminal, ou, uma alternativa mais interativa e visualmente agradável é o notebook Pluto (PLAS et al.,).

Nesse exemplo vamos estar trabalhando com um sistema descrito por:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 2 \\ 1 \end{bmatrix} .C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Definimos então o nosso sistema com essas matrizes e com realimentação estado (SSF).

Considerando também as seguintes matrizes que serão utilizadas para representar conjuntos de restrições:

```
sys = SLIT(
        A = 2×2 Matrix{Float64}:
             1.0  1.0
             0.0  1.0
        B = 2×1 Matrix{Float64}:
             2.0
             1.0
        C = 1×2 Matrix{Float64}:
             1.0  0.0
        feedback = SSF::Feedback = 1
        G = 1×2 Matrix{Float64}:
             1.0  1.0
    )
sys = SLIT(A, B, C, SSF)
```

Figura 5: *Definição do sistema de interesse no ambiente Pluto*

$$X^T = \begin{bmatrix} 0.8 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, \qquad U = \begin{bmatrix} 1.2 \\ -1.5 \end{bmatrix}.$$

Nos quais $Xx \leq \mathbf{1}$ representa as retrições de estado e $Uu \leq \mathbf{1}$ as restrições na entrada.

Podemos então verificar se o poliedro das restrições de estado é invariante com respeito ao sistema que anteriormente definido utilizando a função *is_pinvariant*.

```
ip = is_pinvariant(sys, U, X)

Job 23640258 dispatched
password: mWyKrvHT

---------- Begin Solver Output -----------
Condor submit: 'neos.submit'
Condor submit: 'watchdog.submit'
Job submitted to NEOS HTCondor pool.

termination_status(model) = MathOptInterface.LOCALLY_SOLVED
```

Figura 6: *Resultado Locally Solved do problema de otimização*

Esse exemplo foi escolhido propositalmente, pois o mesmo poliedro de restrições não é invariante se o sistema fosse definido com realimentação de saída. Entretanto podemos utilizar a função que procura um poliedro dentro das restrições que seja invariante. Na figura (7) podemos ver os dois poliedros sobrepostos.

É perceptível que o poliedro não preenche toda o poliedro de restrições dos estados, mas cobre quase toda área, rejeitando apenas uma pequena região do espaço.

3.1 Exemplo: sistema com atraso

Para um sistema com atraso (5) para o qual:

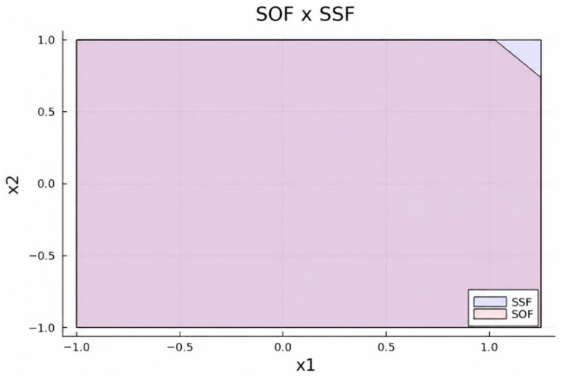


Figura 7: *Poliedro invariante com realimentação de saída*

$$A = \begin{bmatrix} 1.2 & 0.2 \\ -0.4 & 0.6 \end{bmatrix}, \qquad A_d = \begin{bmatrix} -0.3 & -0.2 \\ 0.4 & 0.2 \end{bmatrix},$$

e um conjunto poliedrico $\Omega = \{x \in \mathbb{R}^n : Fx \leq w\}$ com

$$F^T = \begin{bmatrix} -1 & 2 & 1 & -2 \\ -1 & 1 & 1 & -1 \end{bmatrix}, \qquad w^T = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}.$$

Utilizando os programas desenvolvidos verificamos que, para d = 1, o conjunto é invariante, e plotamos na figura (8) um exemplo de trajetória inciando em um dos vértices. Podemos observar que a trajetória vai para origem e não saí dos limites do poliedro.

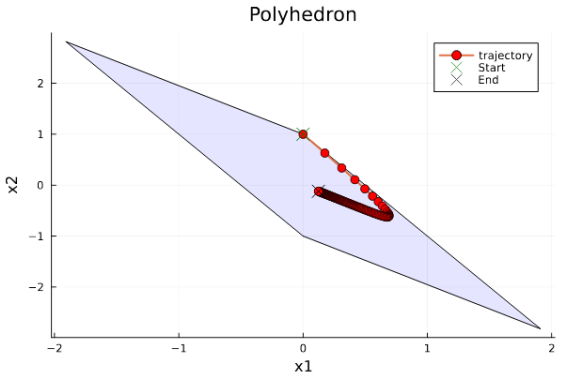


Figura 8: *trajetória dos estados do sistema*

4 Conclusões

Este trabalho apresentou uma biblioteca em *Julia* para o cálculo e a análise de conjuntos invariantes, integrando ferramentas de otimização matemática e

computação poliédrica em um único ambiente. As funcionalidades implementadas permitem definir sistemas dinâmicos, verificar a invariância de conjuntos sob restrições e buscar conjuntos invariantes para vários tipos de sistema, considerando diferentes configurações de realimentação. Os exemplos demonstram a aplicação da biblioteca aos problemas considerados e sua utilidade no estudo de sistemas com atraso na entrada de controle. Como trabalhos futuros, pretende-se ampliar as funcionalidades para outros tipos de conjuntos invariantes e integrar com outras bibliotecas do ecossistema *Julia*.

Referências

BEZANSON, J. et al. Julia: A fresh approach to numerical computing. *SIAM Review*, v. 59, n. 1, p. 65–98, 2017.

BRIÃO, S. L. et al. Output feedback design for discrete-time constrained systems subject to persistent disturbances via bilinear programming. *Journal of the Franklin Institute*, Elsevier, v. 358, n. 18, p. 9741–9770, 2021.

BYRD, R. H.; NOCEDAL, J.; WALTZ, R. A. Knitro: An integrated package for nonlinear optimization. In: PILLO, G. D.; ROMA, M. (Ed.). *Large-Scale Nonlinear Optimization*. Boston: Springer, 2006.

DóREA, C.; OLARU, S. Computation of low complexity invariant sets for time-delay systems: An optimization-based approach. In: *Proc. 18th IFAC Workshop on Control Applications of Optimization (accepted)*. [S.l.: s.n.], 2022. p. 1–6.

DóREA, C. E.; OLARU, S.; NICULESCU, S.-I. Delay-dependent invariance of polyhedral sets for discrete-time linear systems. *IFAC-PapersOnLine*, v. 56, n. 2, p. 3259–3264, 2023. ISSN 2405-8963. 22nd IFAC World Congress.

FOURER, R.; GAY, D. M.; KERNIGHAN, B. W. A modeling language for mathematical programming. *Management Science*, v. 36, n. 5, p. 519–554, 1990.

FUKUDA, K. *cddlib: A C Implementation of the Double Description Method*. [S.l.], 2022. Disponível em: <https://github.com/cddlib/cddlib>.

LEGAT, B. Polyhedral computation. In: *JuliaCon*. [s.n.], 2023. Disponível em: <https://pretalx.com/juliacon2023/talk/JP3SPX/>.

LUBIN, M. et al. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, v. 15, p. 581–589, 2023.

PLAS, F. van der et al. *Pluto.jl: Simple reactive notebooks for Julia*. Disponível em: <https://github.com/JuliaPluto/Pluto.jl>.

The Julia Programming Language. *Functions and types: Multiple dispatch*. 2020. MIT 18.S191, Week 3, Fall 2020. Vídeo publicado em 15 set. 2020. Disponível em: <https://www.youtube.com/watch?v=DXtbaSP_LFI>.

The MathWorks, Inc. *MATLAB*. The MathWorks, Inc. Acesso em: 16 out. 2026. Disponível em: <https://www.mathworks.com/products/matlab.html>.