# PIETOOLS: A Matlab Toolbox for Manipulation and Optimization of Partial Integral Operators

Sachin Shivakumar[1], Amritam Das[2] and Matthew M. Peet[1]

***Abstract*— In this paper, we present PIETOOLS, a MATLAB toolbox for the construction and handling of Partial Integral (PI) operators. The toolbox introduces a new class of MATLAB object, `opvar`, for which standard MATLAB matrix operation syntax (e.g. +, *, ' etc.) is defined. PI operators are a generalization of bounded linear operators on infinite-dimensional spaces that form a *-subalgebra with two binary operations (addition and composition) on the space $\mathbb{R}\times L_2$. These operators frequently appear in analysis and control of infinite-dimensional systems such as Partial Differential Equations (PDE) and Time-delay systems (TDS). Furthermore, PIETOOLS can: declare `opvar` decision variables, add operator positivity constraints, declare an objective function, and solve the resulting optimization problem using a syntax similar to the sdpvar class in YALMIP. Use of the resulting Linear Operator Inequalities (LOI) are demonstrated on several examples, including stability analysis of a PDE, bounding operator norms, and verifying integral inequalities. The result is that PIETOOLS, packaged with SOSTOOLS and MULTIPOLY, offers a scalable, user-friendly and computationally efficient toolbox for parsing, performing algebraic operations, setting up and solving convex optimization problems on PI operators.**

## I. INTRODUCTION

Linear operators on finite-dimensional spaces are defined by matrices. Linear Matrix Inequalities (LMI) provide a computational tool for analysis and control of dynamical systems in such finite dimensional spaces. Recently, the development of Partial Integral Equation (PIE) representations of PDE systems has created a framework for the extension of LMI-based methods to infinite-dimensional systems. The PIE representation encompasses a broad class of distributed parameter systems and is algebraic - eliminating the use of boundary conditions and continuity constraints [1], [2], [3]. Such PIE representations have the form

$$\begin{aligned}\mathcal{H}\dot{\mathbf{x}}(t)+\mathcal{B}_{d1}\dot{w}(t)+\mathcal{B}_{d2}\dot{u}(t)&=\mathcal{A}\mathbf{x}(t)+\mathcal{B}_1 w(t)+\mathcal{B}_2 u(t)\\ z(t)&=\mathcal{C}_1\mathbf{x}(t)+\mathcal{D}_{11}w(t)+\mathcal{D}_{12}u(t),\\ y(t)&=\mathcal{C}_2\mathbf{x}(t)+\mathcal{D}_{21}w(t)+\mathcal{D}_{22}u(t)\end{aligned}$$

where the $\mathcal{H},\mathcal{A},\mathcal{B}_i,\mathcal{C}_i,\mathcal{D}_{ij}$ are Partial Integral (PI) operators and have the form

$$\left(\mathcal{P}\left[\begin{smallmatrix}P, & Q_1\\ Q_2, & \{R_i\}\end{smallmatrix}\right]\begin{bmatrix}x\\ \mathbf{\Phi}\end{bmatrix}\right)(s):=\begin{bmatrix}Px+\int_{-1}^{0}Q_1(s)\mathbf{\Phi}(s)ds\\ Q_2(s)x+\left(\mathcal{P}_{\{R_i\}}\mathbf{\Phi}\right)(s)\end{bmatrix}.$$

where

$$\left(\mathcal{P}_{\{R_i\}}\mathbf{\Phi}\right)(s):=$$

$$R_0(s)\mathbf{\Phi}(s)+\int_{-1}^{s}R_1(s,\theta)\mathbf{\Phi}(\theta)d\theta+\int_{s}^{0}R_2(s,\theta)\mathbf{\Phi}(\theta)d\theta$$

PI operators, which also appear in partial-integro differential equations [4], have been studied in the past [5], [6], [7], extensively. PI operators are integral operators on the joint space of finite dimensional vectors and square integrable functions. Similar to matrices, PI operators are closed under the algebraic operations of addition, concatenation, composition and adjoint. As a result, LMI developed for analysis and control of finite-dimensional systems can be generalized to LOI defined by variables of the `opvar` class. For example, consider the LMI for optimal observer synthesis of singular systems: find $P\succ 0$ and $Z$ such that

$$\begin{bmatrix}-\gamma I & -D_{11}^\top & -(PB+ZD_{21})^\top H\\ (\cdot)^\top & -\gamma I & C_1\\ (\cdot)^\top & (\cdot)^\top & (\cdot)^\top+H^\top(PA+ZC_2)\end{bmatrix}\prec 0$$

where the superscript $^\top$ stands for matrix transpose. This LMI can be generalized to an LOI [3]: Find $\mathcal{P}=\mathcal{P}\left[\begin{smallmatrix}P, & Q\\ Q^T, & \{R_i\}\end{smallmatrix}\right]\succ 0$ and $\mathcal{Z}=\mathcal{P}\left[\begin{smallmatrix}Z_1, & \emptyset\\ Z_2, & \{\emptyset\}\end{smallmatrix}\right]$ such that

$$\begin{bmatrix}-\gamma I & -\mathcal{D}_{11}^* & -(\mathcal{P}\mathcal{B}+\mathcal{Z}\mathcal{D}_{21})^*\mathcal{H}\\ (\cdot)^* & -\gamma I & \mathcal{C}_1\\ (\cdot)^* & (\cdot)^* & (\cdot)^*+\mathcal{H}^*(\mathcal{P}\mathcal{A}+\mathcal{Z}\mathcal{C}_2)\end{bmatrix}\prec 0$$

where the superscript $^*$ stands for operator adjoint. The goal of PIETOOLS is to create a convenient parser for constructing and solving LOI of this form. To this end, PIETOOLS incorporates all elements typically used for constructing LMIs in the commonly used LMI parser YALMIP [8]. Specifically, PIETOOLS can be used to: declare PI operators; declare PI decision variables; manipulate PI objects via addition, multiplication, adjoint, and concatenation; add inequality constraints; set an objective function; and solve an LOI.

Significantly, PIETOOLS also includes scripts for conversion of linear TDS and coupled ODE-PDE models into PIEs. Currently, scripts are also included for stability analysis, $H_\infty$-gain analysis, $H_\infty$-optimal controller synthesis, and $H_\infty$-optimal observer synthesis. These Demo files and PIETOOLS itself are distributed as a free, third party MATLAB toolbox and are available online at [9].

The paper is organized as follows. In Section II, we introduce the standard notation utilized in the paper, followed by formal definition of Partial-Integral (PI) operators in Section III followed by a demonstration of MATLAB implementation of the toolbox in Section IX. In the appendix, we briefly discuss algebraic operations related to PI operators which allow us to solve operator valued tests.

[1] Sachin Shivakumar{sshivak8@asu.edu} and Matthew M. Peet{mpeet@asu.edu} are with School for Engineering of Matter, Transport and Energy, Arizona State University, Tempe, AZ, 85298 USA

[2] Amritam Das{am.das@tue.nl} is with Department of Electrical Engineering, Eindhoven University of Technology

## II. Notation

$\mathbb{S}^m \subset \mathbb{R}^{m\times m}$ is the set symmetric matrices. For a normed space $X$, define $L_2^n[X]$ as the Hilbert space of square integrable $\mathbb{R}^n$-valued functions on $X$ with inner product $\langle x, y\rangle_{L_2} = \int_a^b x(s)^\top y(s)ds$. The Sobolov spaces are denoted $W^{q,n}[X] := \{x \in L_2^n[X] \mid \frac{\partial^k x}{\partial s^k} \in L_2^n[X] \text{ for all } k \le q\}$ with the standard Sobolov inner products. $\mathcal{A}^*$ stands for adjoint of a linear operator $\mathcal{A} : L_2[X] \to L_2[X]$ with respect to standard $L_2$-inner product. For a given inner product space, $Z$, the operator $\mathcal{P} : Z \to Z$ is positive semidefinite (denoted $\mathcal{P} \succcurlyeq 0$) if $\langle z, \mathcal{P}z\rangle_Z \ge 0$ for all $z \in Z$. Furthermore, we say $\mathcal{P} : Z \to Z$ is coercive if there exists some $\epsilon > 0$ such that $\langle z, \mathcal{P}z\rangle_Z \ge \epsilon\|z\|_Z^2$ for all $z \in Z$. The partial derivative $\frac{\partial}{\partial s}\mathbf{x}$ is denoted as $\mathbf{x}_s$. Identity matrix of dimension $n \times n$ is denoted by $I_n$.

## III. PI Operators And PI-operator valued optimization problems

Linear operators mapping between finite-dimensional spaces can be parametrized using matrices. Partial Integral operators (here onwards referred to as PI operators) are a generalization of a linear mapping between infinite-dimensional spaces, specifically a map from $\mathbb{R}^m \times L_2^n \to \mathbb{R}^p \times L_2^q$. These operators are frequently encountered in analysis and control of PDEs or TDSs.

We define two class of PI-operators, 3-PI and 4-PI, where 3-PI operators are a special case of 4-PI operators. As the nomenclature insinuates, 3-PI operators, denoted as $\mathcal{P}_{\{N_i\}} : L_2^m[a,b] \to L_2^n[a,b]$, are parameterized by 3 matrix-valued functions $N_0 : [a,b] \to \mathbb{R}^{n\times m}$ and $N_1, N_2 : [a,b]\times[a,b] \to \mathbb{R}^{m\times n}$ which is a bounded linear operator between two normed spaces $L_2^m[a,b]$ and $L_2^n[a,b]$ endowed with standard $L_2$-inner product.

$$\left(\mathcal{P}_{\{N_i\}}\mathbf{y}\right)(s) = N_0(s)\mathbf{y}(s) + \int_a^s N_1(s,\theta)\mathbf{y}(\theta)d\theta + \int_s^b N_2(s,\theta)\mathbf{y}(\theta)d\theta \tag{1}$$

Similarly, 4-PI operators, parameterized by 4 components, are bounded linear operators between $\mathbb{R}^m \times L_2^n[a,b]$ and $\mathbb{R}^p \times L_2^q[a,b]$.

$$\mathcal{P}\left[\begin{smallmatrix} P, & Q_1 \\ Q_2, & \{R\}\end{smallmatrix}\right]\begin{bmatrix}x\\ \mathbf{y}\end{bmatrix}(s) = \begin{bmatrix} Px + \int_a^b Q_1(s)\mathbf{y}(s)ds \\ Q_2(s)x + \mathcal{P}_{\{R_i\}}\mathbf{y}(s)\end{bmatrix} \tag{2}$$

where $P : \mathbb{R}^m \to \mathbb{R}^p$, $Q_1 : [a,b] \to \mathbb{R}^{p\times n}$, $Q_2 : [a,b] \to \mathbb{R}^{q\times m}$ and $\mathcal{P}_{\{R_i\}} : L_2^n[a,b] \to L_2^q[a,b]$.

These operators frequently appear in control-related applications for linear TDS or coupled ODE-PDE systems. Linear TDS or coupled ODE-PDE systems with boundary conditions can be rewritten using PI operators (see [1]). Stability test of such a system gives rise to an operator-valued feasibility test, as shown below.

***Example** 1 (Feasibility):*
Test for the stability of a coupled ODE-PDE system, whose dynamics are governed by the equation, in PI format,

$$\mathcal{H}\dot{\mathbf{x}} = \mathcal{A}\mathbf{x} \tag{3}$$

can be posed as an operator-valued feasibility test, shown below.

$$\text{Find, } \mathcal{P} \succ 0, \ s.t.$$
$$\mathcal{A}^*\mathcal{P}\mathcal{H} + \mathcal{H}^*\mathcal{P}\mathcal{A} \preccurlyeq 0$$

If there exists a self-adjoint coercive PI operator $\mathcal{P}$, which satisfies the given constraints, then the system governed by Eq. (3), is stable.

Another application of interest is finding the $H_\infty$-norm of a coupled ODE-PDE system. This can be posed as an optimization problem minimizing the $L_2$-gain bound from inputs to outputs.

***Example** 2 (Optimization):*
Finding $H_\infty$-norm, $\gamma$, of a coupled ODE-PDE system whose dynamics are governed by the equation in PI format shown below

$$\begin{aligned}\mathcal{H}\dot{\mathbf{x}} &= \mathcal{A}\mathbf{x} + \mathcal{B}u,\\ y &= \mathcal{C}\mathbf{x} + \mathcal{D}u,\end{aligned} \tag{4}$$

can be posed as the following optimization problem.

$$\begin{aligned}&\text{minimize } \gamma, \ s.t.\\ &\mathcal{P} \succ 0,\\ &\begin{bmatrix} -\gamma I & \mathcal{D}^* & \mathcal{B}^*\mathcal{P}\mathcal{H} \\ \mathcal{D} & -\gamma I & \mathcal{C} \\ \mathcal{H}^*\mathcal{P}\mathcal{B} & \mathcal{C}^* & \mathcal{A}^*\mathcal{P}\mathcal{H} + \mathcal{H}^*\mathcal{P}\mathcal{A}\end{bmatrix} \preccurlyeq 0\end{aligned}$$

Although the examples provided here are control-oriented, PIETOOLS is capable of solving other operator-valued feasibility or convex optimization problems, as described in Section IX.

## IV. Declaration and Manipulation of `opvar` Objects

PIETOOLS introduces the structured `opvar` class of MATLAB object, each element of which consists of a PI operator $\mathcal{P}\left[\begin{smallmatrix} P, & Q_1 \\ Q_2, & \{R_i\}\end{smallmatrix}\right] : \mathbb{R}^m \times L_2^n[a,b] \to \mathbb{R}^p \times L_2^q[a,b]$. The structural elements of an `opvar` object are listed in Table I. The elements $P$, $Q1$, $Q2$, $R.R0$, $R.R1$, and $R.R2$ are themselves of the `pvar` class of polynomial introduced in the MULTIPOLY toolbox. Note that the MULTIPOLY toolbox is included in PIETOOLS toolbox, along with a modified version of SOSTOOLS. For this reason, the PIETOOLS path should take precedence over any path containing a preexisting version of MULTIPOLY or SOSTOOLS. In the solvers distributed with PIETOOLS, this is ensured by executing the Matlab command `addpath(genpath('.'))` from a file within the PIETOOLS directory.

`opvar` variables can be defined in MATLAB in two ways. The first method is directly using the `opvar` command, which is used to define opvar objects with known properties. The other method is by declaring an `opvar` decision variable - as described in Section V.

The command `opvar` takes in string inputs and initializes them as symbolic opvar objects with default properties. These properties can be modified using standard MATLAB assignment. The following code snippet demonstrates a simple example.

```
>> opvar P1 P2;
```

TABLE I: List of properties in `opvar` class and their description

| Property | Value |
|---|---|
| var1, var2 | A `pvar` object |
| P | A matrix with dimensions $p \times m$ |
| Q1 | A matrix-valued `pvar` object in var1 with dimensions $p \times n$ |
| Q2 | A matrix-valued `pvar` object in var1 with dimensions $q \times m$ |
| R.R0 | A matrix-valued `pvar` object in var1 with dimensions $q \times n$ |
| R.R1, R.R2 | A matrix-valued `pvar` object in var1 and var2 with dimensions $q \times n$ |
| I | A vector with entries `[a,b]` |
| dim | A matrix with values `[p,m;q,n]` |

```
>> P1.I = [0 1];
>> P1.P = rand(2,2); P1.Q1 = rand(2,1);
```

The above code snippet would create two `opvar` variables `P1` and `P2` with default values. Next, the interval of `P1` is changed to $[0, 1]$. Finally, components $P$ and $Q_1$ are reassigned with random matrices of stated dimensions. This makes `P1` a PI operator mapping $\mathbb{R}^2 \times L_2[a, b] \to \mathbb{R}^2$.

In addition to defining a new class, PIETOOLS overloads MATLAB operators such as `+`, `*` and `'` to simplify manipulation of PI-operators. Set of PI operator is a *-subalgebra with binary operations (addition and composition), i.e., it is a *-ring with the involution also being an associative subalgebra. This allows operations such as addition, composition, concatenation, and adjoint to be performed in a manner similar to matrices. All these operations result in another PI operator and can be performed in MATLAB.

### A. Addition Of 4-PI Operators

Two 4-PI operators $\mathcal{P}\left[\begin{smallmatrix} A, & B_1 \\ B_2, & \{C_i\} \end{smallmatrix}\right]$ and $\mathcal{P}\left[\begin{smallmatrix} L, & M_1 \\ M_2, & \{N_i\} \end{smallmatrix}\right]$ can be added, if they have same dimensions, to obtain another 4-PI operator $\mathcal{P}\left[\begin{smallmatrix} P, & Q_1 \\ Q_2, & \{R_i\} \end{smallmatrix}\right]$ where

$$P = A + L, Q_i = B_i + M_i, R_i = C_i + N_i.$$

In MATLAB, two `opvar` class objects `P1` and `P2` can be added by using the MATLAB operator + as shown below.

```
>> P1+P2
```

### B. Composing Two 4-PI Operators

Two 4-PI operators $\mathcal{P}\left[\begin{smallmatrix} A, & B_1 \\ B_2, & \{C_i\} \end{smallmatrix}\right]$ and $\mathcal{P}\left[\begin{smallmatrix} P, & Q \\ Q_2, & \{R_i\} \end{smallmatrix}\right]$ can be composed if their inner dimensions match. The composition results in a new 4-PI operator $\mathcal{P}\left[\begin{smallmatrix} \hat{P}, & \hat{Q}_1 \\ \hat{Q}_2, & \{\hat{R}_i\} \end{smallmatrix}\right]$, where

$$\hat{P} = AP + \int_a^b B_1(s)Q_2(s)\mathrm{d}s,$$

$$\hat{Q}_1(s) = AQ_1(s) + B_1(s)R_0(s) + \int_s^b B_1(\eta)R_1(\eta, s)\mathrm{d}\eta + \int_a^s B_1(\eta)R_2(\eta, s)\mathrm{d}\eta,$$

$$\hat{Q}_2(s) = B_2(s)P + C_0(s)Q_2(s) + \int_a^s C_1(s, \eta)Q_2(\eta)\mathrm{d}\eta + \int_s^b C_2(s, \eta)Q_2(\eta)\mathrm{d}\eta,$$

$$\hat{R}_0(s) = C_0(s)R_0(s),$$

$$\hat{R}_1(s, \eta) = B_2(s)Q_1(\eta) + C_0(s)R_1(s, \eta) + C_1(s, \eta)R_0(\eta) + \int_a^\eta C_1(s, \theta)R_2(\theta, \eta)\mathrm{d}\theta + \int_\eta^s C_1(s, \theta)R_1(\theta, \eta)\mathrm{d}\theta + \int_s^b C_2(s, \theta)R_1(\theta, \eta)\mathrm{d}\theta,$$

$$\hat{R}_2(s, \eta) = B_2(s)Q_1(\eta) + C_0(s)R_2(s, \eta) + C_2(s, \eta)R_0(\eta) + \int_a^s C_1(s, \theta)R_2(\theta, \eta)\mathrm{d}\theta + \int_s^\eta C_2(s, \theta)R_2(\theta, \eta)d\theta + \int_\eta^b C_2(s, \theta)R_1(\theta, \eta)\mathrm{d}\theta.$$

In MATLAB, the composition of two `opvar` objects `P1` and `P2` is performed by using `*`.

```
>> P1*P2
```

### C. Adjoint Of A 4-PI Operator

A 4-PI operator $\mathcal{P}\left[\begin{smallmatrix} \hat{P}, & \hat{Q}_1 \\ \hat{Q}_2, & \{\hat{R}_i\} \end{smallmatrix}\right]$ is the adjoint of the 4-PI operator $\mathcal{P}\left[\begin{smallmatrix} P, & Q_1 \\ Q_2, & \{R_i\} \end{smallmatrix}\right]$, with respect to $\mathbb{R} \times L_2$-inner product, if

$$\hat{P} = P^\top, \qquad \hat{R}_0(s) = R_0^\top(s),$$
$$\hat{Q}_1(s) = Q_2^\top(s), \qquad \hat{R}_1(s, \eta) = R_2^\top(\eta, s),$$
$$\hat{Q}_2(s) = Q_1^\top(s), \qquad \hat{R}_2(s, \eta) = R_1^\top(\eta, s).$$

The adjoint of an `opvar` class object `P1` can be computed using the following MATLAB syntax.

```
>> P1'
```

### D. Concatenation Of 4-PI Operators

Horizontal and vertical concatenation of `opvar` class objects, `P1` and `P2`, with compatible dimensions can be done using the following two commands, respectively.

```
>> [P1 P2]
>> [P1; P2]
```

## V. Declaring `opvar` Decision Variables

Predefined `opvar` objects can be input using the syntax as described in Section IV. In addition, PIETOOLS can be used to set up and solve optimization problems with `opvar` decision variables. Before declaring `opvar` variables, the optimization problem structure must be initialized. This process is inherited from the SOSTOOLS toolbox and consists of the following syntax.

```
>>T = sosprogram([s,th],gam);
```

Here `s`, `th`, and `gam` are `pvar` objects. The structured object `T` carries an accumulated list of variables and constraints and must be passed whenever an additional variable or constraint is declared. The commands `sos_opvar` and `sos_posopvar` both declare `opvar` objects with unknown parameters. The latter function adds the constraint that the associated PI operator be positive. The syntax for both functions are listed as follows.

### A. sos_opvar

```
>> [T,P] = sos_opvar(T,dim,I,s,th,deg);
```

Indefinite opvar decision variables can be defined using the sos_opvar command. This function has six required inputs:

1) An empty or partially complete problem structure T to which to add the variable;
2) A length two vector I=[a,b], indicating the spatial domain of the operator;
3) Two pvar objects s and th, corresponding to the pvar objects declared in sosprogram when T was initialized;
4) A $2\times2$ matrix dim=[p m; q n], indicating the dimension of domain and range of the operator; Note that when $q=n=1$, the decision variable is a matrix.
5) A length 3 vector deg=[d1,d2,d3] which control the degrees of the polynomials in P. The value d1 is the highest degree of s in the polynomials P.Q1, P.Q2 and P.R.R0. The values d2 and d3 correspond to the highest degree of s and th, respectively, in the polynomials P.R.R1 and P.R.R2.

sos_opvar returns an opvar object P which is the 4-PI operator $\mathcal{P}\left[\begin{smallmatrix}\texttt{P.P}, & \texttt{P.Q1}\\ \texttt{P.Q2}, & \{\texttt{P.R.Ri}\}\end{smallmatrix}\right] : \mathbb{R}^m \times L_2^n[a,b] \to \mathbb{R}^p \times L_2^q[a,b]$ and the problem structure T to which the variable has been appended. For more details on construction of an opvar object check Appendix A.

### B. sos_posopvar

```
>> [T,P] = sos_posopvar(T,dim,I,s,th,deg);
```

Positive semi-definite opvar decision variables can be defined using the sos_posopvar command. This function has six required inputs:

1) An empty or partially complete problem structure T to which to add the variable;
2) A length two vector I=[a,b], indicating the spatial domain of the operator;
3) Two pvar objects s and th, corresponding to the pvar objects declared in sosprogram when T was initialized;
4) A $2\times1$ vector dim=[m; n], indicating the dimension of domain and range of the operator. Note that when $n=0$, this becomes a standard positive matrix variable.
5) An array deg=[d1,d2,d3] which control the degrees of the polynomials in P. The highest degree in the polynomials P.Qi are all max(d1, d2+d3+1). The highest degree in P.R.R0 is 2d2. The highest degrees of s and th in P.R.R1 and P.R.R2 are all max(d1+d2, 2d2+d3+1).

sos_posopvar returns an opvar object P which is a self-adjoint 4-PI operator $\mathcal{P}\left[\begin{smallmatrix}\texttt{P.P}, & \texttt{P.Q1}\\ \texttt{P.Q2}, & \{\texttt{P.R.Ri}\}\end{smallmatrix}\right]$ on $\mathbb{R}^m \times L_2^n[a,b]$ and the problem structure T to which the variable has been appended. The functions P.P, P.Q1, P.Q2, P.R.Ri are constrained such that P is a positive semidefinite operator. For more details on construction of a positive opvar object check Appendix B.

## VI. Constraining opvar Class Objects And Solving An Optimization Problem

In addition to declaring opvar objects with unknown variables by using sos_opvar or sos_posopvar, the user can add equality constraints or operator positivity constraints to a problem structure.

### A. sos_opineq

```
>> T = sos_opineq(T,P);
```

sos_opineq adds an operator inequality constraint of the form $\mathcal{P}\left[\begin{smallmatrix}\texttt{P.P}, & \texttt{P.Q1}\\ \texttt{P.Q2}, & \{\texttt{P.R.Ri}\}\end{smallmatrix}\right] \geq 0$ to a problem structure T. The function has two required inputs: a problem structure T to which to append the constraint; and an opvar structure P which is constrained to be positive semidefinite in the augmented problem structure T returned by the function.

### B. sos_opeq

```
>> T = sos_opeq(T,P);
```

sos_opeq adds an operator equality constraint of the form $\mathcal{P}\left[\begin{smallmatrix}\texttt{P.P}, & \texttt{P.Q1}\\ \texttt{P.Q2}, & \{\texttt{P.R.Ri}\}\end{smallmatrix}\right] = 0$ to a problem structure T. The function has two required inputs: a problem structure T to which to append the constraint; and an opvar structure P which is constrained to be zero in the augmented problem structure T returned by the function.

For example, the constraint P2=P1 can be imposed by using the command

```
>> T = sos_opeq(T,P2-P1);
```

where P1 and P2 are opvar objects.

### C. Defining the objective via sossetobj

Solving optimization problems using PIETOOLS, as introduced in Section V, may require specification of an objective. This is done by using sossetobj function inherited from SOSTOOLS.

```
>>T = sossetobj(T,gam);
```

sossetobj adds the objective function gam to the problem structure T. There are two necessary inputs: The scalar pvar object gam object which is to be the minimized and the problem structure T to which the objective is to be added .

### D. Solving the optimization problem

Once all elements of the optimization problem have been added to the problem structure T, the problem can be solved by using the function sossolve, inherited from SOSTOOLS and which requires an instance of SeDuMi available in the Matlab path.

```
>>T = sossolve(T);
```

sossolve has a single input which is an 'unsolved' problem structure T. The function returns the problem structure in a 'solved' state. Data on the solution can now be obtained from the problem structure. For more details on sossetobj and sossolve we refer to the most recent SOSTOOLS documentation in [10].

### E. sosgetsol_opvar

After execution of sossolve, optimal values of the real-valued decision variables may be extracted from the problem structure using the command sosgetsol, as described in the SOSTOOLS documentation. To extract a feasible opvar decision variable, the sosgetsol_opvar function may be used.

```
>>P = sosgetsol_opvar(T,P);
```

sosgetsol_opvar This function takes necessary inputs: a solved optimization problem structure T and the name of the opvar decision variable P whose solution is to be retrieved. The function returns an opvar object with no decision variables. This object may be manipulated further or used in the definition of a new problem structure. problem structures in the 'solved' state cannot be re-used.

TABLE II: List of functions for opvar class and their description

| Function | Description |
|---|---|
| opvar | Creates default opvar object with given names |
| + | Adds two opvar objects |
| * | Composes two opvar objects |
| ' | Transposes an opvar object |
| sos_opvar | Returns opvar variable of given dimensions |
| sos_posopvar | Returns a self-adjoint, opvar variable which is constrained to be positive semidefinite |
| sos_opeq | Takes an input opvar variable, P, and adds the constraint P = 0 |
| sos_opineq | Takes an input opvar variable, P, and adds the constraint P $\geq$ 0 |
| sosgetsol_opvar | Returns the value of an opvar decision variable after solving the optimization problem |

## VII. PIETOOLS scripts for analysis and control PDEs and Systems with Delay

As described in the Section I, PDEs and Delay Systems admit PIE representations which can be used to test for stability, find the $H_\infty$-norm or design the $H_\infty$-optimal observers and controllers - See [1]. PIETOOLS includes the scripts solver_PIETOOLS_PDE and solver_PIETOOLS_TDS, which take input parameters as described in the header of the file and constructs to the corresponding PIE representation using the script setup_PIETOOLS_PDE or setup_PIETOOLS_TDS. Once converted to PIE form, the solver file calls one of the following executives based on the user input.

1) executive_PIETOOLS_stability: This executive is called if the user sets stability=1 in the solver file. This tests if the PDE or TDS in PIE form is stable.
2) executive_PIETOOLS_Hinf_gain: This executive is called if the user sets Hinf_gain=1 in the solver file. The executive returns a bound on the $H_\infty$-gain of the PDE or TDS in PIE form.
3) executive_PIETOOLS_Hinf_estimator: This executive is called if the user sets Hinf_estimator=1 in the solver file. The executive searches for an $H_\infty$-optimal observer for the PDE or TDS in PIE form.
4) executive_PIETOOLS_Hinf_controller: This executive is called if the user sets Hinf_control=1 in the solver file. The executive searches for an $H_\infty$-optimal controller for the PDE or TDS in PIE form.

For example, consider the stability test for a linear PDE system using PIETOOLS.

***Demonstration** 1 (Stability):* Solution, $u$, of a tip-damped wave equation is governed by

$$u_{tt}(s,t) = u_{ss}(s,t), \quad u(0,t) = 0, \quad u_s(1,t) = -ku_t(1,t).$$

With a simple change of variable, this can be converted to two PDEs first-order differential in time

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_t (s,t) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_s (s,t),$$
$$u_2(0,t) = 0, \; u_1(1,t) = -ku_2(1,t)$$

where $u_1 = u_s$ and $u_2 = u_t$.

To test for stability of this system, we seclare the parameters a,b,A1,B as follows in the file solver_PIETOOLS_PDE and set stability=1.

```
a=0; b=1;
A1 = [0,1;1,0]; B = [0,1,0,0;0,0,k,1]
```

## VIII. Solving Optimization Problems Using PIETOOLS - A Summary

As discussed in section III, PIETOOLS can solve feasibility tests or optimization problems involving opvar decision variables and equality/inequality constraints. This section provides a brief outline of the steps necessary for setting up and solving such an optimization problem.

Recall from Example 2 that to find $H_\infty$-norm of the system defined by Eqn. (4) we may solve the following optimization problem.

$$\text{minimize } \gamma, \; s.t.$$
$$\mathcal{P} \succ 0,$$
$$\begin{bmatrix} -\gamma I & \mathcal{D}^* & \mathcal{B}^*\mathcal{P}\mathcal{H} \\ \mathcal{D} & -\gamma I & \mathcal{C} \\ \mathcal{H}^*\mathcal{P}\mathcal{B} & \mathcal{C}^* & \mathcal{A}^*\mathcal{P}\mathcal{H} + \mathcal{H}^*\mathcal{P}\mathcal{A} \end{bmatrix} \preccurlyeq 0$$

To solve this optimization problem using PIETOOLS, the following steps are necessary.

1) Define pvar objects.

```
pvar s,th,gam;
```

2) Initialize a problem structure.

```
T = sosprogram([s,th],gam);
```

3) Define relevant opvar data-objects.

```
opvar A,B,C,D,H;
A=..;B=..;C=..;,D=..;H=..;X=[0,1];
```

4) Add decision variables to the problem structure.

```
[T,P] = sos_posopvar(T,dim,X,s,th);
```

5) Add constraints to the problem structure.

```
D = [-gam*I   D'      B'*P*H;
      D      -gam*I   C;
      H'*P*B  C'      A'*P*H+H'*P*A];
T = sosopineq(T,D);
```

6) Add an objective to the problem structure (minimize gam).

```
T = sossetobj(T,gam);
```

7) Solve the completed problem structure.

```
T = sossolve(T);
```

8) Extract the solution to the 'solved' problam.

```
P_s = sosgetsol_opvar(T,P);
```

## IX. Demonstrations of PIETOOLS Usage

In this section, a few simple examples are presented to demonstrate the use of PIETOOLS. Apart from control-related applications, described in previous sections, users can set up and solve other convex optimization problems that involve opvar variables. For instance, one can: find a tight upper bound on the induced norm of a PI operator - operators which appear in e.g. the backstepping transformation [11] and input-output maps of non-linear ODEs [12]. Such bounds on the induced norm are obtained as follows.

***Demonstration** 2 (Operator norm):* Find the $L_2$ induced operator norm for Volterra integral operator

$$(\mathcal{A}x)(s) = \int_0^s x(t)dt.$$

The operator $\mathcal{A}$ is a 4-PI operator with $R_1 = 1$ and all other elements 0. The $L_2$ induced operator norm is defined as $min\{\sqrt{\gamma} \mid \langle \mathcal{A}x, \mathcal{A}x\rangle \leq \gamma \langle x, x\rangle, \forall x \in L_2[a,b]\}$. The corresponding optimization problem is

$$\min \gamma, s.t.$$
$$\mathcal{A}^*\mathcal{A} \leq \gamma.$$

Start by defining relevant pvar and opvar objects.

```
>> pvar s th gam;
>> opvar A; A.R.R1 = 1; A.I=[0,1]
```

Next, initialize a problem structure with $s$, $th$ and $gam$ as pvar objects. $gam$ is the objective to be minimized.

```
>> prog = sosprogram([s,th],[gam]);
>> prog = sossetobj(prog,gam);
```

Next, add the opvar inequality constraint using the sos_opineq function.

```
>> prog = sos_opineq(prog, A'*A-gam);
```

Finally, the problem can be solved and solution extracted using the following commands.

```
>> prog = sossolve(prog);
>> Gam = sosgetsol(prog, gam);
>> disp(sqrt(Gam));
ans =
    0.6366
```

The numerical value of .6366 obtained from PIETOOLS can be compared to the analytical value of the induced norm of this operator norm which is known to be $\frac{2}{\pi} \approx 0.6366$.

PIETOOLS can also be used to provide certificates of positivity for integral inequalities.

***Demonstration** 3 (Poincare's Constant):* Poincare's Inequality states that there exists a constant $C$ such that for every function $u \in W_0^{1,p}(\Omega)$ (where $W_0^{1,p}(\Omega)$ is the Sobolev space of zero-trace functions) we have that

$$\|u\|_{L_p(\Omega)} \leq C\|\nabla u\|_{L_p(\Omega)}$$

where $1 \leq p < \infty$ and $\Omega$ is a bounded set. This can be rewritten as an optimization problem.

$$\min C, s.t.$$
$$\langle u, u\rangle - C\langle u_s, u_s\rangle \leq 0.$$

For $p = 2$ and $\Omega = [0,1]$, it known that for functions $u \in W_0^{2,2}(\Omega)$ with the boundary conditions $u(0) = u(1) = 0$ the smallest $C = 1/\pi^2$. Numerical calculation of this constant $C$ can be reformulated as a PI optimization problem as follows.

$$\min C, s.t.$$
$$\mathcal{H}^*\mathcal{H} - C\mathcal{H}_2^*\mathcal{H}_2 \leq 0$$
$$(\mathcal{H}u)(s) = \int_a^s (s\theta - \theta)u(\theta)d\theta + \int_s^b (s\theta - s)u(\theta)d\theta$$
$$(\mathcal{H}_2u)(s) = \int_a^s \theta u(\theta)d\theta + \int_s^b (\theta - 1)u(\theta)d\theta$$

The set up and solution of this PI optimization problem using PIETOOLS is as follows.

```
pvar s t C; opvar H H2;
H.I=[0,1]; H.R.R1=s*t-t; H.R.R2=s*t-s;
H2.I=[0,1]; H2.R.R1 = t; H2.R.R2 = t-1;
prog = sosprogram([s,t],C);
prog = sossetobj(prog,C);
prog = sos_opineq(prog, H'*H-C*H2'*H2);
prog = sossolve(prog);
```

When implemented, this code returns a smallest bound of $\sqrt{C} = .3183 \approx \frac{1}{\pi}$

## X. Conclusions

In this paper, we have provided a guide to the new MATLAB toolbox PIETOOLS for manipulation and optimization of PI operators. We have provided details on declaration of PI operator objects, manipulation of PI operators, declaration of PI decision variables, addition of operator equality and inequality constraints, solution of PI optimization problems, and extraction of feasible operators. We have demonstrated the practical usage of PIETOOLS, including scripts for analysis and control of PDEs and systems with delay, as well as bounding operator norms and proving integral inequalities. These examples and descriptions illustrate both the syntax of available features and the necessary components of any PIETOOLS script. Finally, we note that PIETOOLS is still under active development. Ongoing efforts focus on identifying and balancing the degree structures in sos_opineq.


## Acknowledgment

This work was supported by Office of Naval Research Award N00014-17-1-2117 and National Science Foundation under Grants No. 1739990 and 1935453.


## References


[1] S. Shivakumar, A. Das, S. Weiland, and M. Peet, "A generalized LMI formulation for input-output analysis of linear systems of ODEs coupled with PDEs," in *Proceedings of the IEEE Conference on Decision and Control*, 2019.

[2] M. M. Peet, S. Shivakumar, A. Das, and S. Weiland, "Discussion paper: A new mathematical framework for representation and analysis of coupled PDEs," *3rd IFAC Workshop on Control of Systems Governed by Partial Differential Equations CPDE 2019*, vol. 52, no. 2, pp. 132 – 137, 2019.
[3] A. Das, S. Shivakumar, S. Weiland, and M. Peet, "$H_\infty$ optimal estimation for linear coupled PDE systems," in *Proceedings of the IEEE Conference on Decision and Control*, 2019.
[4] A. Smyshlyaev and M. Krstic, "Closed-form boundary state feedbacks for a class of 1D partial integro-differential equations," *IEEE Transactions on Automatic Control*, vol. 49, no. 12, pp. 2185–2202, 2004.
[5] S. Bergman, *Integral operators in the theory of linear partial differential equations*. Springer, 2013.
[6] J. Appell, A. Kalitvin, and P. Zabrejko, *Partial Integral Operators and Integro-Differential Equations: Pure and Applied Mathematics*. CRC Press, 2000.
[7] I. Gohberg, S. Goldberg, and M. A. Kaashoek, *Classes of linear operators*. Birkhäuser, 2013, vol. 63.
[8] J. Löfberg, "YALMIP : A toolbox for modeling and optimization in MATLAB," in *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
[9] M. Peet and S. Shivakumar, "PIETOOLS for Networks with Delay," https://codeocean.com/capsule/0447940/tree.
[10] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. Parrilo, "SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB," *arXiv preprint arXiv:1310.4716*, 2013.
[11] M. Krstic and A. Smyshlyaev, *Boundary control of PDEs: A course on backstepping designs*. Siam, 2008, vol. 16.
[12] J. V. Feijoo, K. Worden, and R. Stanway, "Associated linear equations for Volterra operators," *Mechanical Systems and Signal Processing*, vol. 19, no. 1, pp. 57–69, 2005.
[13] S. Shivakumar, A. Das, and M. M. Peet, "PIETOOLS: A MATLAB toolbox for manipulation and optimization of partial integral operators," *arXiv preprint arXiv:1910.01338*, 2019.

## APPENDIX

### A. Construction of `opvar` variables

```
>> [T,P] = sos_opvar(T,dim,I,s,th,deg);
```

Suppose the inputs to the function are `T`, `I=[a,b]`, `s`, `th`, `dim =[p m; q n]` and `deg =[d1 d2 d3]`. The function first creates two vector of monomials $Z_1$ and $Z_2$. $Z_1$ is vector of monomials in `s` up to the degree `d1`. $Z_2$ has monomials in both `s` and `th` such that highest degrees of `s` and `th` are restricted to `d2` and `d3`, respectively. Then, matrix variables $T_0 : \mathbb{R}^{p\times m}$, $T_1 : \mathbb{R}^{p\times jn}$, $T_2 : \mathbb{R}^{q\times jm}$, $S_0 : \mathbb{R}^{q\times jn}$ and $S_1, S_2 : \mathbb{R}^{q\times kn}$ are created and added to the list of variables in `T` where $j$ and $k$ are lengths of $Z_1$ and $Z_2$. The components of `opvar` object, `P`, are assigned the values as shown below.

$$\begin{aligned}
&\texttt{P.P} = T_0, && \texttt{P.R.R0} = S_0 Z_1 \otimes I_n,\\
&\texttt{P.Q1} = T_1 Z_1 \otimes I_n, && \texttt{P.R.R2} = S_1 Z_2 \otimes I_n,\\
&\texttt{P.Q2} = T_2 Z_1 \otimes I_m, && \texttt{P.R.R2} = S_2 Z_2 \otimes I_n,\\
&\texttt{P.I} = [a,b], \quad \texttt{P.var1} = \texttt{s}, \quad \texttt{P.var2} = \texttt{th}. && (5)
\end{aligned}$$

### B. Construction of `posopvar` variables

```
>> [T,P] = sos_posopvar(T,dim,I,s,th,deg);
```

Suppose the inputs to the function are `T`, `I=[a,b]`, `s`, `th`, `dim =[m; n]` and `deg =[d1 d2 d3]`. Similar to `sos_opvar`, the function `sos_posopvar` first creates two vector of monomials $Z_{d1}$ and $Z_{d2}$. $Z_{d1}$ is vector of monomials in `s` up to the degree `d1`. $Z_{d2}$ has monomials in both `s` and `th` such that highest degrees of `s` and `th` are restricted to `d2` and `d3`, respectively. Then, matrix variables $T_1 : \mathbb{R}^{m\times m}$, $T_2 : \mathbb{R}^{m\times(j+2k)n}$ and $T_3 : \mathbb{R}^{(j+2k)n\times(j+2k)n}$ are created and added to the list of variables in `T` where $j$ and $k$ are lengths of $Z_1$ and $Z_2$. Further, the constraint $\begin{bmatrix} T_1 & T_2 \\ T_2^\top & T_3 \end{bmatrix} \geq 0$ is added to `T`. Then functions $Z_0$, $Z_1$ and $Z_2$ are created as follows

$$Z_0 = \begin{bmatrix} \sqrt{g} Z_{d1} \otimes I_n \\ 0 \\ 0 \end{bmatrix}, Z_1 = \begin{bmatrix} 0 \\ \sqrt{g} Z_{d2} \otimes I_n \\ 0 \end{bmatrix},$$

$$Z_2 = \begin{bmatrix} 0 \\ 0 \\ \sqrt{g} Z_{d2} \otimes I_n \end{bmatrix}$$

where $g(s) = s(L-s)$ or $g = 1$. Finally, the `posopvar` object is created using the composition formula

$$\mathcal{P}\left[\begin{smallmatrix} P, & Q_1 \\ Q_2, & \{R_i\} \end{smallmatrix}\right] = \mathcal{P}\left[\begin{smallmatrix} I, & 0 \\ 0, & \{Z_i\} \end{smallmatrix}\right]^* \mathcal{P}\left[\begin{smallmatrix} T_1, & T_2 \\ T_2^\top, & \{T_3, 0, 0\} \end{smallmatrix}\right] \mathcal{P}\left[\begin{smallmatrix} I, & 0 \\ 0, & \{Z_i\} \end{smallmatrix}\right].$$