# PIETOOLS_2020a: How to get started?

Sachin Shivakumar, Amritam Das, and Matthew Peet

July 2, 2020

**Abstract**

You are reading this file means that you are interested in working with PI operators and using PIETOOLS_2020a. Congratulations! You are very cool and tech-savvy. This file gives you a step by step guide on how to make your first program in PIETOOLS and execute it.

## 1 Before You Start

Before you start, **make sure** that you have checked the following setting in your computer

1. MATLAB with version 2014a or newer.
2. An SDP solver. Sedumi is included in the installation script and can be obtained from this link.
3. `PIETOOLS_2020a.zip` is downloaded, and unzipped from this link.
4. The file `pietools_install.m` has been run in your MATLAB. If you didn't bother with steps 2 and 3, then this install file will take care of them.

## 2 Start With An Example

To make your first program in PIETOOLS_2020a, the best way is to run one of the demo files located in `GetStarted_DOCS_DEMOS` folder. Specifically, the file `Getting_Started_DEMO.m` includes the example used in this Getting Started Document. The remainder of this section will give everything essential you need to know to start your journey with PIETOOLS.

To do that, let us consider an example where the objective is to construct the PIE representation and verify the stability of a simple reaction-diffusion equation.

**Example 2.1** *Consider a diffusion reaction equation in an interval* $[0,1]$ *as follows:*

$$\frac{\partial x(s,t)}{\partial t} = \frac{\partial^2 x(s,t)}{\partial s^2} + \lambda x(s,t), \tag{1}$$

*with Dirichlet boundary conditions,* $x(0,t) = x(1,t) = 0$.

*It can be shown that, when* $\lambda < \pi^2 \approx 9.8696$*, the system is exponentially stable.*

Let us implement a test to verify this stability property in PIETOOLS. The code to be implemented in PIETOOLS has three parts: a) specify the model, b) Convert the PDE to a PIE, and c) setup and solve the stability problem as an LPI in PIETOOLS.

## 2.1 Specify the model

To specify the considered model along with the boundary conditions, the following items must be included

```
>> n0=0;n1=0;n2=1;
>> lambda = 9;
>> A0=lambda; A1=0; A2=1;
>> a=0;b=1;
>> B=[1 0 0 0; 0 1 0 0];
```

Let us go line by line to understand the above code.

- `>> n0=0;n1=0;n2=1;`

  Here, number of all the state variables are specified. There are three categories of states. In particular, `n0, n1, n2` are the number of state variables that have no differentiation with respect to $s$, with restrictions of first order differentiability with respect to $s$ and with restrictions on second order differentiability with respect to $s$ respectively. In the model (1), $u(s,t)$ is maximally twice differentiable. Hence, `n2=1`.

  The need to specify `n0, n1, n2` is the least cool and most confusing thing about PIETOOLS. We have worked hard to get rid of this step or make it easier, but have failed. Sorry!

- `>> lambda = 9;`

  Here, we specify the value of $\lambda$ for which the stability of (1) is to be tested. Note, for $\lambda = 9$, the system (1) is exponentially stable.

- `>> A0=lambda; A1=0; A2=1;` Here, we specify the coefficients of PDE. Similar to assigning the state variables, `A0; A1; A2` assigns coefficients corresponding to each of the three categories of states.

- `>> a=0;b=1;`

  This defines the extremum points of the spatial interval where (1) is defined.

- `>> B=[1 0 0 0; 0 1 0 0];`

  Here, we define the boundary conditions. The boundary conditions have the following format:

$$B \begin{bmatrix} x(0,t) \\ x(1,t) \\ \frac{\partial x(0,t)}{\partial s} \\ \frac{\partial x(1,t)}{\partial s} \end{bmatrix} = 0.$$

## 2.2 Construct and View the PIE Representation

Constructing the PIE representation of a PDE by hand is not for beginners. Fortunately, PIETOOLS_2020a can do this for you with a well-developed script. Just enter the following two lines of code, and you are done!

```
>> pvar s theta;
>> convert_PIETOOLS_PDE;
```

Let us go line by line to understand the above code.

- `>> pvar s theta;`

  Here, using `pvar`, we assign the first and second independent variables of the polynomial functions that appear in the definition of a PI operator. We have chosen them to be `s theta` and they are used in the converters and executables. DO NOT CHANGE THEM!

- `>> convert_PIETOOLS_PDE;`

Easy as PIE! But wait... Maybe you want to see the PIE you have created. Simple PIEs like this one have the form

$$\mathcal{T}\dot{\mathbf{u}}(t) = \mathcal{A}\mathbf{u}(t).$$

where $\mathbf{u}(t,s) = \partial_s^2 x(t,s)$ (Because you used `n2=1`). This looks pretty abstract. To get a better feel for what is going on, you can view the operators $\mathcal{A}$ and $\mathcal{T}$ by entering the following lines of code

```
>> Aop
>> Top
```

PIETOOLS will display your PI operators in a visually appealing manner:

```
Aop=

      [] | []
      -----------
      [] | Aop.R

Aop.R=

    [1] | [9*s*theta-9*theta] | [9*s*theta-9*s]
Top=

      [] | []
      -----------
      [] | Top.R

Top.R=

    [0] | [s*theta-theta] | [s*theta-s]
```

In this case, we can take the pieces of the `Aop` and `Top` and write the PIE directly as follows

$$\overbrace{\int_0^s \underbrace{\theta(s-1)}_{\texttt{Top.R.R1}}\dot{u}(t,\theta)ds+\int_s^1 \underbrace{s(\theta-1)}_{\texttt{Top.R.R2}}\dot{u}(t,\theta)ds}^{\mathcal{T}\dot{\mathbf{u}}(t)}$$
$$=\underbrace{\overbrace{1}^{\texttt{Aop.R.R0}}u(t,s)+\int_0^s \overbrace{9\theta(s-1)}^{\texttt{Aop.R.R1}}u(t,\theta)ds+\int_s^1 \overbrace{9s(\theta-1)}^{\texttt{Aop.R.R2}}u(t,\theta)ds}_{\mathcal{A}\mathbf{u}(t)}$$

The main advantage of the PIE representation is that the effect of the boundary conditions has been inserted directly into the dynamics. This means we can use optimization algorithms for things like stability analysis.

## 2.3 Solve The Stability Problem

Now that you have your PIE, its time to do something with it. Let's test whether this PIE is stable. This requires us to set up and solve an LPI. Setting up and solving an LPI is pretty easy in PIETOOLS. It only takes the following 4 lines of code:

```
>> prog = sosprogram([s; theta]);
>> [prog, Pop] = poslpivar(prog, [0 ,1],[0 1]);
>> Pop = Pop+.0001;
>> options.psatz = 1;
>> prog = lpi_ineq(prog,-[Top'*Pop*Aop+Aop'*Pop*Top], options);
>> prog = sossolve(prog);
```

Let us go line by line to understand the above code.

- `>> prog = sosprogram([s; theta]);`

  Here, we initialize a new optimization program called `prog` to solve the stability problems. The independent variables related to this program are the same for the PI operators. In this case, they are `[s; theta]`.

- `>> [prog, Pop] = poslpivar(prog, [0 ,1],[0 1]);`

  ```
  >> Pop = Pop+.0001;
  >> options.psatz = 1;
  >> prog = lpi_ineq(prog,-[Top'*Pop*Aop+Aop'*Pop*Top], options);
  ```

  The above two lines of codes declares the LPIs corresponding to the stability problem. Recall the PIE has the form $\mathcal{T}\dot{\mathbf{x}}(t) = \mathcal{A}\mathbf{x}(t)$. The stability analysis amounts to verifying that there exists a positive PI operator $\mathcal{P} \succ 0$ such that $-(\mathcal{T}^{'}\mathcal{P}\mathcal{A} + \mathcal{A}^{'}\mathcal{P}\mathcal{T}) \succ 0$. In the above two lines of code

  1. $\mathcal{T}, \mathcal{A}$ are given by `Top` and `Aop` respectively. These two PI operators are automatically generated by the subroutine `convert_PIETOOLS_PDE` based on the user-defined model in Section 2.1. The positive PI operator $\mathcal{P}$ is declared as `Pop`. This is an unknown variable.
  2. To perform algebraic operations on PI operators like addition, composition, adjoint and concatenation, PIETOOLS includes simple commands. Here for addition, composition and transpose, these commands are `+` and `*` and `'` respectively.
  3. To declare a positive PI operator and add it as an unknown PI variable for the program, you can use `poslpivar`. It takes three inputs. Apart from the specific program `prog`, the second input to `poslpivar` is the number of states associated to the ODE (in our case 0) and associated to PDEs (in our case 1). The third input is the interval on which the unknown PI operator is positive. In this case, this interval is the interval on which the PDE is defined, i.e. `[0 1]`.
  4. `Pop = Pop+.0001` is added to make the `Pop` strictly positive.
  5. `options.psatz = 1` is enabled as an additional functionality to the PI operator that enables Putinar positivestellensatz. In this way, the PI operator is only restricted to be negative in the domain [0 1].
  6. To associate the program with additional LPIs, you can use `lpi_ineq`. Apart from the specific program `prog`, `lpi_ineq` takes LPIs as a positivity constraint. In our case, that is `-[Top'*Pop*Aop+Aop'*Pop*Top]`.

- `>> prog = sossolve(prog);`

  Here, we solve the LPIs declared in the the specific program `prog` by using the command `sossolve`.

### 2.4 Display The Result

Now to visualize the result, as an option, you may like to add the following code

```
>> if norm(prog.solinfo.info.feasratio-1)<=.1
        disp('The System of equations was successfully solved.');
   else
        disp('The System is of equations not solved.');
   end
```

Here, we verify whether the search for positive `Pop` was successful or not by running a diagnostic test on the feasibility and numerical errors. In this case, for `lambda = 9`, the displayed message will be `The System of equations was successfully solved.` This means, the system (1) is stable. If you now change the `lambda` to 10, and execute the file again, you will see that the system is unstable.

## 3 Learning More About PI Operators

Many of the operations mentioned in this example automated in PIETOOLS by various executive files. Moreover, more functionalities can be added for defining the LPIs. To know more about them, go over the demo files added with PIETOOLS or learn about a specific PIETOOLS routine by simple using the `help` command.