PIETOOLS: How to get started?

S. Shivakumar A. Das D. Braghini D. Jagt Y. Peet
M. Peet

November 16, 2024

Abstract

In this document, we give a step-by-step guide on how to code your first program in PIETOOLS and execute it. The topics covered are setting up a PDE system using the parser format, converting to and interpreting the PIE representation, testing internal stability, and interpreting results of the LPI test.

1 Before You Start

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

1. MATLAB with version 2019a or newer.
2. An SDP solver. SeDuMi is included in the installation script and can be obtained from `this link`.
3. `PIETOOLS` is downloaded from `this link`.

2 Start With An Example

To make your first program in PIETOOLS, the best way is to run one of the demo files located in `PIETOOLS_demos` folder. 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 transport equation.

Example 2.1 *Consider the PDE defined as follows:*

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

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

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

Let us implement a test to verify the internal stability property in PIETOOLS. The code implemented in PIETOOLS has three parts: a) specify the model, b) Convert the PDE to a PIE, and c) Set up 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

```
>> x = state('pde'); pvar s t; lam = 9;
>> pde = sys();
>> dynamics = [diff(x,t)==diff(x,s,2)+lam*x];
>> BC = [subs(x,s,0)==0; subs(x,s,1)==0];
>> pde = addequation(pde,[dynamics;BC]);
```

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

- `>> x = state('pde'); pvar s t; lam = 9;`

  This line defines a symbolic-type variable '`x`' that can then be freely manipulated to define equations, as shown below. We also define the spatial (`s`) and temporal (`t`) variables to be used in operations such as differentiation (`diff`) and substitution (subs). Lastly, we specify the parameter λ to be 9 and store it in a variable (`lam`).

- `>> pde = sys();`

  Next, we create a container-type object (`pde`) to store the information related to the PDE system. Typically, this object stores the equations and certain auxiliary information such as the states, domain, inputs, outputs, etc.

- `>> dynamics = [diff(x,t)==diff(x,s,2)+lam*x];`

  Equations are simply defined by using standard symbolic-type manipulation of the predefined variables ('`x`', '`s`', '`lam`', and '`t`'). Here, we use '`diff(x,t)`' to define partial derivative in time $\partial_t x(t,s)$. Likewise, $\partial_s^2 x(t,s)$ could be declared using '`diff(x,s,2)`', where now the third argument specifies the order of differentiation. To finally define the equation, we use the equality symbol '`==`', multiplication symbol '`*`', and addition '`+`' to obtain `diff(x,t)==diff(x,s,2)+lam*x`, which stands for $\partial_t x = \partial_s^2 x + \lambda x$ and is stored in '`dynamics`' variable.

- `>> BC = [subs(x,s,0)==0; subs(x,s,1)==0];`

  Similarly, we define and store boundary conditions using the '`subs`' function which substitutes '`s`' by 0 and 1 to define the two boundary conditions $x(t,0) = 0$ and $x(t,1) = 0$, respectively.

- `>> pde = addequation(pde,[dynamics;BC]);`

  Once all the equations are defined, we collect them and add them to the '`pde`' system object that was created earlier using '`addequation`' function as shown above.

You can verify if the PDE has been defined correctly by displaying the 'pde' object by typing '`pde`' in the command window. You would get an output as shown below.

```
    >> pde

pde =


  dt x(t,s) = ds^2 x(t,s) + 9 * x(t,s);

  0 = - x(t,0);
  0 = - x(t,1);
```

2.2 Convert to PIE representation

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

```
    >> pie = convert(pde,'pie');
```

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

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

where $u(t,s) = \partial_s^2 x(t,s)$. 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

```
    >> pie.A
    >> pie.T
```

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

```
pie.A=

      [] | []
      -----------
      [] | pie.A.R

pie.A.R=

    [1] | [9*s*theta-9*theta] | [9*s*theta-9*s]
pie.T=
```

```
    [] | []
    -----------
    [] | pie.T.R
```

```
pie.T.R=

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

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

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

The PIE representation's main advantage is that the boundary conditions' effect 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, it's 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 one line of code:

```
[prog, Pop] = lpiscript(pie, 'stability', 'veryheavy');
```

This instructs PIETOOLS to run a stability test for the given 'pie.params' with very heavy settings (very high number of decision variables in the parametrization of the Lyapunov function and derivative of the Lyapunov function).

2.3.1 Interpret The Result

To interpret the result, you can investigate the output from SeDuMi, which should look something like the text below.

```
     --- Executing Primal Stability Test ---
- Parameterizing Positive Lyapunov Operator using specified options...
- Constructing the Negativity Constraint...
- Enforcing the Negativity Constraint...
  - Using an Equality constraint...
- Solving the LPI using the specified SDP solver...
Size: 9041   156
```

```
SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, Adaptive Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 156, order n = 160, dim = 9042, blocks = 4
nnz(A) = 23522 + 0, nnz(ADA) = 23940, nnz(L) = 12048
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            8.12E+00 0.000
  1 :   7.46E-07 2.68E+00 0.000 0.3298 0.9000 0.9000   1.00  1  1  1.2E+01
  2 :   6.87E-07 9.66E-01 0.000 0.3609 0.9000 0.9000   1.00  1  1  4.2E+00
  3 :   2.72E-07 3.08E-01 0.000 0.3186 0.9000 0.9000   1.00  1  1  1.3E+00
  4 :   9.61E-08 8.90E-02 0.000 0.2891 0.9000 0.9000   1.00  1  1  3.8E-01
  5 :   3.25E-08 3.05E-02 0.000 0.3428 0.9000 0.9000   1.00  1  1  1.3E-01
  6 :   3.25E-08 6.04E-03 0.081 0.1979 0.9000 0.0000   1.00  1  1  5.5E-02
  7 :   3.25E-08 2.64E-03 0.320 0.4372 0.9000 0.0000   1.00  1  1  3.4E-02
  8 :   2.22E-08 1.72E-03 0.055 0.6501 0.9000 0.9000   1.00  1  1  2.3E-02
  9 :   2.22E-08 7.67E-04 0.267 0.4467 0.9000 0.0000   1.00  1  1  1.4E-02
 10 :   1.59E-08 4.61E-04 0.040 0.6011 0.9000 0.4148   1.00  1  1  8.9E-03
 11 :   8.39E-09 2.08E-04 0.000 0.4515 0.9000 0.7878   1.00  1  1  4.0E-03
 12 :   2.98E-09 5.79E-05 0.000 0.2782 0.9072 0.9000   1.00  1  1  1.2E-03
 13 :   5.88E-10 2.41E-05 0.000 0.4162 0.9000 0.8496   1.00  1  1  4.6E-04
 14 :   1.73E-10 7.92E-06 0.000 0.3286 0.9000 0.8333   1.00  1  1  1.5E-04
 15 :   5.79E-11 2.46E-06 0.000 0.3107 0.9013 0.9000   1.00  1  1  4.8E-05
 16 :   2.33E-11 9.28E-07 0.000 0.3772 0.9000 0.9013   1.00  1  2  1.8E-05
 17 :   9.00E-12 3.61E-07 0.000 0.3894 0.9000 0.8575   1.00  1  2  7.1E-06
 18 :   2.91E-12 1.22E-07 0.000 0.3380 0.9000 0.8901   1.00  2  2  2.4E-06
 19 :   1.21E-12 5.18E-08 0.000 0.4238 0.9000 0.8326   0.99  4  4  1.0E-06
 20 :   3.99E-13 1.71E-08 0.000 0.3302 0.9003 0.9000   0.99  5  5  3.4E-07
 21 :   1.49E-13 6.61E-09 0.000 0.3868 0.9000 0.8483   1.00 10 11  1.3E-07
 22 :   5.49E-14 2.84E-09 0.000 0.4296 0.9000 0.8715   1.00 32  9  5.2E-08
 23 :   2.13E-14 9.49E-10 0.000 0.3341 0.9000 0.8003   1.00 18 17  1.9E-08
 24 :   1.03E-14 3.65E-10 0.000 0.3840 0.9042 0.9000   1.00 12 12  7.4E-09

iter seconds digits       c*x                b*y
 24      3.0  12.5  0.0000000000e+00  1.0309210258e-14
|Ax-b| =   3.8e-09, [Ay-c]_+ =   1.3E-09, |x|=  1.6e+01, |y|=  3.2e+03

Detailed timing (sec)
   Pre          IPM          Post
1.087E+00    4.497E+00    1.420E-01
Max-norms: ||b||=9.000000e-06, ||c|| = 0,
Cholesky |add|=4, |skip| = 6, ||L.L|| = 2.80222e+06.

Residual norm: 3.8336e-09

         iter: 24
```

```
feasratio: 0.9999
     pinf: 0
     dinf: 0
   numerr: 0
   timing: [1.0870 4.4970 0.1420]
  wallsec: 5.7260
   cpusec: 3.3750
```

Here, we verify whether the search for a positive operator `Pop` certifying stability was successful or not by running a diagnostic test on the feasibility and numerical errors. In the case of `lam = 9`, the `feasratio` is almost 1, and the primal infeasibility flag (`pinf`) and dual infeasibility flag (`dinf`) are both zero. This indicates that the stability LPI was successfully solved, and thus the system (1) is stable. If you run the same codes but now setting the value of `lam` to 10, you will find that in that case the stability test fails, as the system is unstable.

3 Learning More About PI Operators

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 simply using the `help` command.