PIETOOLS 2021b: Getting Started with PDEs

Sachin Shivakumar, Amritam Das, Declan Jagt, and Matthew Peet

December 29, 2021

Abstract

The fact that you are reading this file means that you are interested in working with PI operators and using PIETOOLS_2021b. 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, please **make sure** of the following:

- You have MATLAB version 2014a or newer installed.
- You have an SDP solver installed. The solver SeDuMi is included in the installation script, and can also be obtained from `this link`.
- You have downloaded `PIETOOLS_2021b.zip` (from `control.asu.edu/pietools/`), and unzipped the file.
- You have run the file `pietools_install.m` in MATLAB. If you didn't bother with steps 2 and 3, then this install file will take care of them.

2 Let's get started

Welcome to PIETOOLS! This file will give you a brief demonstration of how you can use this toolbox, and get you started on your PIE journey.

Wait, what's a PIE?

A Partial Integral Equation, or PIE, is an alternative representation for a large class of differential systems, such as Partial Differential Equations (PDEs). In a PIE, partial differential operators are replaced by Partial Integral (PI) operators, which allow for much easier analysis of your system. For example, take a run-of-the-mill transport equation:

$$\begin{aligned} &\dot{u}(t,s) = \partial_s u(t,s) && s \in [0,1], \quad t \geq 0 \\ &u(t,0) = 0 \end{aligned}$$

Though it may not be immediately obvious, this PDE, like any PDE, imposes 3 types of constraints on the state u:

1. An evolution equation: $\dot{u}(t,s) = \partial_s u(t,s)$
2. Boundary conditions: $u(t,0) = 0$
3. Continuity constraints: $\partial_s u \in L_2[0,1]$ (i.e. u must be at least once differentiable)

The presence of these latter two types of constraints tends to make analysis of PDEs quite difficult, with many methods relying on ad hoc steps to be taken to take them into account. However, if we consider an alternative state $\hat{u} = \partial_s u$, then this state is not restricted by any continuity or boundary constraints. Moreover, we can retrieve our PDE state u from this alternative state using the fundamental theorem of calculus, and exploiting the boundary conditions:

$$u(t,s) = u(t,0) + \int_0^s \partial_\theta u(t,\theta) d\theta = \int_0^s \hat{u}(t,\theta) d\theta$$

We can therefore represent our PDE in an equivalent manner as

$$\int_0^s \dot{\hat{u}}(t,\theta) d\theta = \hat{u}(t,s) \qquad s \in [0,1], \quad t \geq 0,$$

which is a PIE! Note that the state in this system is only constrained by the actual evolution equation, allowing us to study this system (and hence our transport equation) without having to account for additional constraints.

Well, what if I have a different PDE?

Any (well-defined) linear PDE can be converted to an equivalent PIE. And the best part is: you can use PIETOOLS to do it! For example, consider a simple heat equation:

$$\begin{aligned} &\dot{u}(t,s) = \partial_s^2 u(t,s) && s \in [0,1], \quad t \geq 0 \\ &u(t,0) = \partial_s u(t,0) = 0 \end{aligned}$$

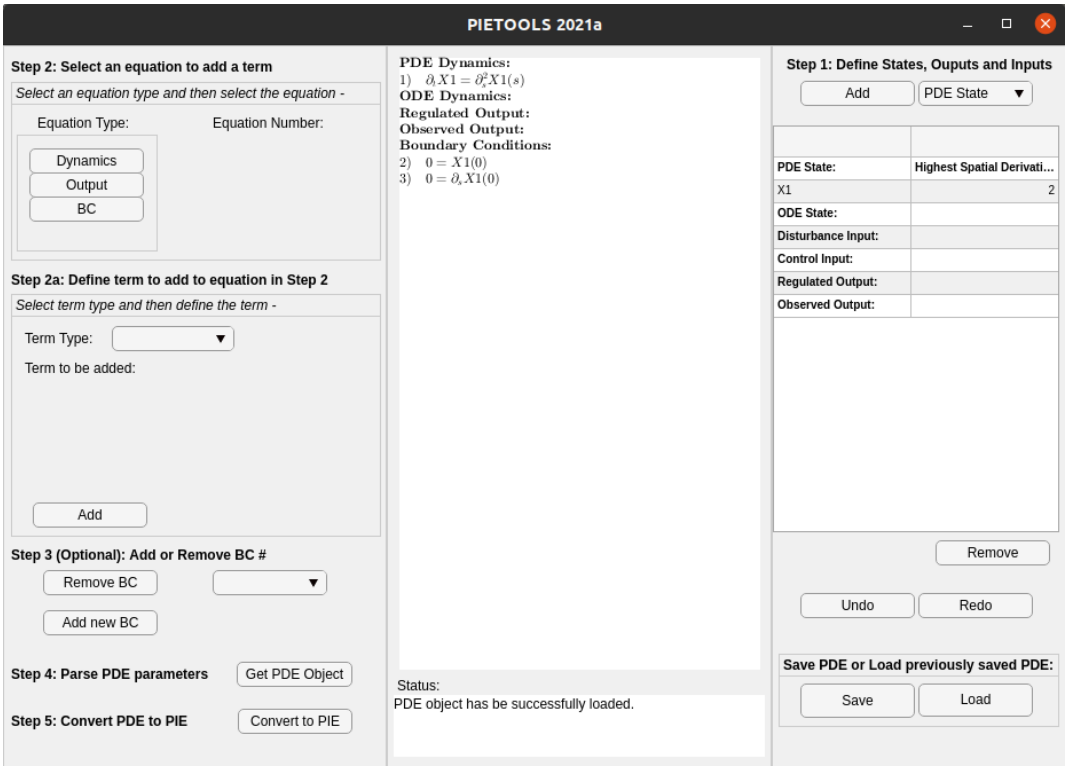
This system can be easily specified in PIETOOLS using the app `PIETOOLS_PDE_GUI`, so feel free to do so. However, we have also implemented this system for you, saved as a file `Heat_Eq.mat` in the folder `/Examples_PDE_GUI/Demos`. You can open this file in one of two ways:

1. By loading it directly into the GUI. This can be achieved by running the GUI (`PIETOOLS_PDE_GUI`), and using the "Load" button at the bottom right of the window. Locate the file `Examples_GUI_PDE/Demos/Heat_Eq.mat`, press "open", and you're done!

2. Using the command line. To this end, define `root` as the location where you installed PIETOOLS (the parent folder containing all the PIETOOLS files and folders), and run:

```
>> app = PIETOOLS_2021b;
>> load(fullfile(root,'PIETOOLS_examples/Examples_PDE_GUI/Demos/Heat_Eq.mat'));
```

If everything went well, you should see a window like this:



You can see that the PDE described in the GUI corresponds to our heat equation. Then, to describe this system in a format PIETOOLS can work with, we can press the button "Get PDE Object" at the bottom left of the window. This will add a MATLAB strucutre `PDE_GUI` to your workspace, describing the heat equation. More interesting though, is the conversion to a corresponding PIE. This can be performed pressing the "Convert to PIE" button at the bottom left of the window, and will add a structure `PIE_GUI` to your workspace. This structure will describe the PIE representation of our heat equation, which should be of the form

$$\mathcal{T}\dot{\hat{\mathbf{u}}} = \mathcal{A}\hat{\mathbf{u}},$$

where $\hat{\mathbf{u}} = \partial_s^2 u$ is our PIE state. To see what the PIE looks like, we can extract the values of operators $\mathcal{T}$ and $\mathcal{A}$ from the `PIE` structure, by running

```
>> Top = PIE_GUI.T
>> Aop = PIE_GUI.A
```

which will produce an output

```
Top=

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

Top.R=

    [0] | [s-theta] | [0]
Aop=

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

Aop.R=

    [1] | [0] | [0]
```

Here, `Top` and `Aop` are "opvar" objects, describing PI operators in PIETOOLS. From the values of `Top.R` and `Aop.R` we can determine that our PIE looks like

$$\overbrace{\int_0^s \underbrace{[s-\theta]}_{\texttt{Top.R.R1}} \dot{\hat{\mathbf{u}}}(t,\theta)d\theta}^{\mathcal{T}\dot{\hat{\mathbf{u}}}(t)} = \overbrace{\underbrace{1}_{\texttt{Aop.R.R0}} \hat{\mathbf{u}}(t,s)}^{\mathcal{A}\hat{\mathbf{u}}(t)}.$$

It is easy to verify that $\big(\mathcal{T}\hat{\mathbf{u}}\big)(t,s) = u(t,s)$, and $\big(\mathcal{A}\hat{\mathbf{u}}\big)(t,s) = \partial_s^2 u(t,s)$, retrieving our heat equation. So, we can use the GUI to specify any (well-posed) PDE, and immediately convert it to a corresponding PIE!

So now what?

Now that we have a PIE, we can use it to analyze our PDE. For analysis of PDEs, you can use the `PIETOOLS_PDE` script in the root directory. For example, if we want to test whether our PDE is stable, open the script, and uncomment the line saying `stability=1`. This tells the script that you want to test stability of the submitted system, which can also be done by specifying `stability=1` in the command line. Now, to make sure the script can actually find your problem, rename your PIE structure to the standard name: `PIE=PIE_GUI;`. Also, make sure the line `PDE = examples_PDE_library_PIETOOLS();` is commented, to avoid overwriting your PDE with a predefined example. Finally, uncomment the line `settings_PIETOOLS_light`, and comment all the other `settings_PIETOOLS` lines. This will limit the effort PIETOOLS will use to test stability, decreasing computation time, but reducing the accuracy. Now, if you run the script you will likely get an output like:

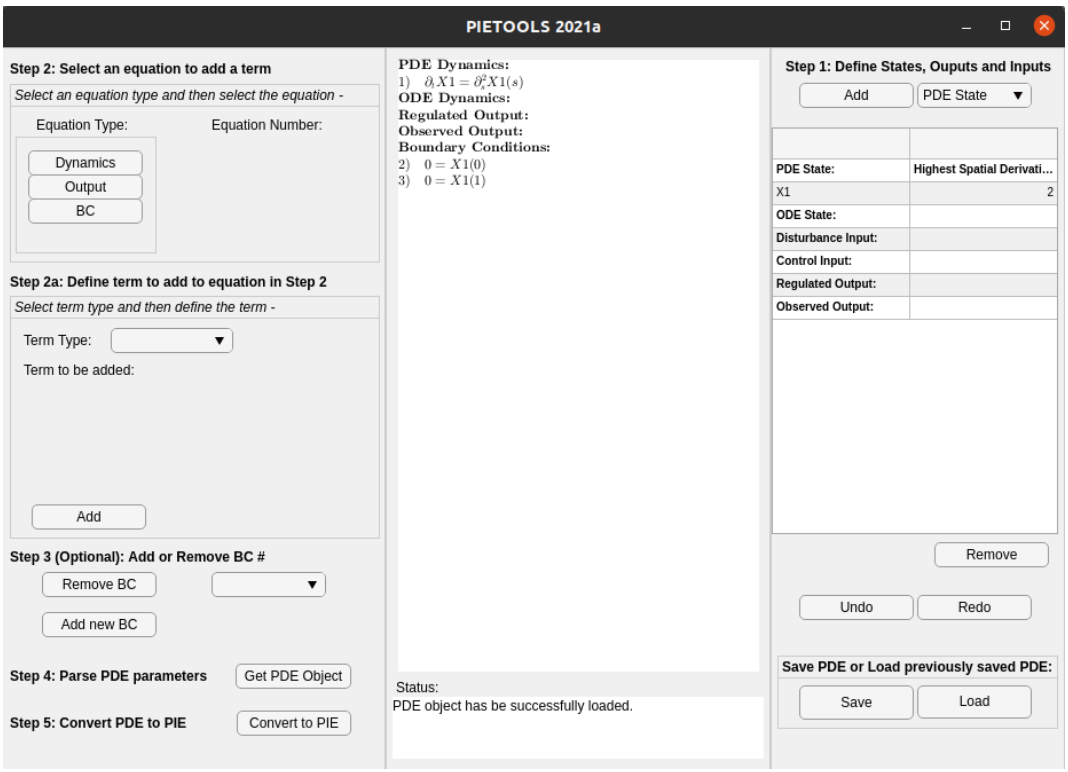
```
Residual norm: 2e-06

         iter: 9
    feasratio: -1.0000
         pinf: 1
         dinf: 0
       numerr: 0
       timing: [0.0043 0.0654 0.0043]
      wallsec: 0.0740
       cpusec: 0.2300

The System of equations was not solved.
```

This message tells us that PIETOOLS is unable to verify stability. In particular, the value `pinf: 1` suggests that, using the "light" settings, PIETOOLS considers the stability problem to be infeasible. You can try playing around with adjusting the settings, for example uncommenting `settings_PIETOOLS_heavy` instead of `settings_PIETOOLS_light`, but you will find that PIETOOLS cannot tell you with certainty that this system is stable. Therefore, let's consider a slightly different system, concerning a heat equation with Dirichlet conditions on both of the boundaries:

$$\begin{aligned} &\dot{u}(t,s) = \partial_s^2 u(t,s) && s \in [0,1], \quad t \geq 0 \\ &u(t,0) = u(t,1) = 0 \end{aligned}$$

To specify this system, we can simply adjust our old heat equation in the GUI, replacing the boundary condition $\partial_s u(t,0) = 0$ with $u(t,1) = 0$. To this end, use the drop-down menu under step 3 (in the bottom-left of the window) to specify boundary condition 3, and press "Remove BC". This will remove the undesired boundary condition. Then, press "Add new BC", which will add a new boundary condition "0=0" to the system. Note that we cannot adjust the left-hand side of this boundary condition, which will always have to be 0. This means that we will have to set the right-hand side of this condition equal to "X1(1)", so that "0=X1(1). To do this, move to step 2 in the top-right, and select "BC". Then, use the drop-down menu to the right of this option to specify the boundary condition we wish to add a term to: number 3. Next, in step 2a, use the drop-down menu to specify that we wish to add a "Standard" (non-integral) term. Use the new drop down menu to specify the

state we wish to add to our boundary conditions (our only state "X1"), and select "1" to the right of this state to indicate we want to evaluate the state at the upper boundary. Since we don't want to take a derivative of the state (at the boundary), nor do we want to multiply it with any coefficient, simply press "Add" to add the specified term to boundary condition 3. Then, you should have a system that looks like:



Now, to test stability for this example, use "Get PDE Object" to add a PDE structure `PDE_GUI` to your workspace. Rename this structure `PDE=PDE_GUI` to make sure `PIETOOLS_PDE` can find your PDE, and run the script with the same options as before. This will produce an output that looks like:

```
Residual norm: 1.6474e-08

        iter: 9
    feasratio: 0.9991
        pinf: 0
        dinf: 0
      numerr: 1
      timing: [0.0045 0.0713 0.0011]
     wallsec: 0.0770
      cpusec: 0.1100

The System of equations was successfully solved. However, Double-check the precision.
```

PIETOOLS has no problem verifying stability of this system, which you can also check using heavier settings. Feel free to play around adjusting your PDE and see how this may affect stability. For example, try implementing the PDE $\dot{u}(t,s) = \partial_s^2 u(t,s) + \lambda u$ for some value λ, with the same boundary conditions. It can be shown that this system is stable for $\lambda < \pi^2 \approx 9.8696$; a bound you can verify up to high accuracy using PIETOOLS.

Is that it?

Of course that's not it! There are tons of things you can do with PIETOOLS. You can couple your PDE to an ODE, and introduce inputs and outputs. You can compute H_∞-optimal controllers and observers, and construct the corresponding closed-loop system. You can work with NDSs, DDFs, DDEs, PDEs, or focus purely on PIs. Just check the manual to see all the cool things you can do with PIETOOLS. Also, if you're looking for inspiration, check out the example libraries, or look at the other demo files. There's plenty to do, so have fun!