Solving transient advection-diffusion-reaction equations by orthogonal-collocation method with special constraints via `Pyomo.dae`

Guoxiang Grayson Tong[1]

[1]Ph.D. student, Department of Applied and Computational Mathematics and Statistics

1 Abstract

Numerically solving advection-diffusion-reaction systems has been an active research field for decades. In general, robust numerical approximations should be both well-behaved numerically, *i.e.* stability, fast convergence and *etc*, and also highly related to the real physics that the model represents. Unfortunately, most numerical techniques have their cons, either showing unsatisfactory numerical performance or violating certain physical/analytical laws. This project aims to further dive into this field of study by the combination of optimization tools and finite difference collocation methods. For certain scenarios, constraints like monotonicity, positivity of the solution will be added the simulation as to formulate a constrained optimization problem, which is usually not easily handled by the numerical scheme itself. In addition, the recent developed `pyomo.dae` will be utilized to realize the numerical examples. By the results, we found .....

2 Introduction

`Python`-based `pyomo.dae` [8], developed in recent years, provides convenient approaches to handle numerical optimization problems involving differential and algebraic equations. Explicitly, `pyomo.dae` prevails most existing optimization solvers by its support of PDE/ODE/DAE of arbitrary order, canonical or non-canonical forms, arbitrary domains and classical syntax of `Python`. Based on the knowledge learned in class and homework, this project will further investigate the power of `pyomo.dae` in solving PDE and PDE-constrained optimization problems.

The PDEs of interest are the unsteady advection-diffusion-reaction equations:

$$\begin{aligned}
\frac{\partial \phi}{\partial t} + \mathbf{b}\cdot\nabla\phi - \nu\Delta\phi + c\phi &= f && \text{in } \Omega\times[0,T]\\
\nabla\cdot\mathbf{b} &= 0 && \text{in } \Omega\times[0,T]\\
\phi &= g && \text{on } \Gamma_D\times[0,T]\\
\nabla\phi\cdot\mathbf{n} &= h && \text{on } \Gamma_N\times[0,T]\\
\phi &= \phi_0 && \text{on } \Omega\times[0]
\end{aligned}$$

We can also write the first PDE as a compact operator form: $\frac{\partial \phi}{\partial t} + \mathcal{L}(\phi) = f$. This is also known as the scalar transport equations, where the scalar ϕ could be the concentration of a chemical, temperature, density, *etc*. The meanings of the above parameters are as: $\mathbf{b}$: the incompressible advection field, ν: the diffusivity constant, c: the reaction constant, f: the source/sink. Besides, the problem is defined on the domain $\Omega\in\mathbb{R}^n$ with Dirichlet and Neumann boundaries Γ_D, $\Gamma_N\subset\partial\Omega\in\mathbb{R}^{n-1}$, respectively. In addition, to make this problem well-defined, we also need proper boundary conditions g, h and initial condition ϕ_0. This

PDE is indeed a conservation law, which describes how a conserved quantity is transported in space time via macroscopic (momentum forces, advection) and microscopic (molecular diffusion, reaction) effect.

Numerically solving the above PDE can be achieved by many approaches: the finite element method [5], the finite difference method [9], the finite volume method [7], the Lattice-Boltzmann method [4] and the orthogonal collocation method [1] [8] that `pyomo.dae` unitizes to realize dynamic optimization. In general, robust numerical approximation of the above system should produce results satisfying both analytical and physical properties [5]. For example, the elliptic operator requires the global satisfaction of the maximum principle [2] [5]:

$$\text{If } \mathcal{L}\phi \geq 0 \text{ in } \Omega, \text{ then,}$$
$$\min_{\bar{\Omega}} \phi = \min_{\partial\Omega} \phi$$

For non-negative forcing, the above directly leads to the non-negative constraint:

$$\min_{\bar{\Omega}} \phi = 0$$

This condition is also closely related to the real physics, *i.e.*, we are not expecting negative density and concentration. Furthermore, we still need the numerical solution to be monotone, in the cases of no reaction and forcing, since in certain applications, oscillatory results clearly violates the second law of the thermodynamics. However, most numerical methods are not oscillation-free because of various reasons [6]. Oscillations usually bring nonphysical over/undershoots, especially in the presence of of internal and boundary layers. In this project, we intend to resolve the aformentioned issues by optimization techniques. We classify the satisfactions of "discrete maximum principle, non-negativity and monotonicity" as **special constraints**.

3 Model development

Enforcing special constraints to the steady advection-diffusion system has been well-studied in the past by finite-element based variational multiscale method [5]. This project aims to extend this idea to finite-difference based orthogonal collocation methods via `pyomo.dae` (upon finishing, the code of this project will be opened to public as the footnote below[1]). Further, transient problem and additional reaction term will be considered. Namely, we are interested in the following discrete optimization problem:

$$\begin{aligned}
&\text{Find } \phi_h \text{ in } \bar{\Omega}_h \times [0,T], \text{ s.t.} \\
&\frac{\partial \phi_h}{\partial t} + \mathcal{L}(\phi_h) = f \\
&\phi_h|_{\Gamma_D} = g_h \\
&\phi_h|_{t=0} = \phi_{0,h} \\
&\min_{\bar{\Omega}_h} \phi_h = 0 \\
&\min_{\bar{\Omega}_h} \phi_h = \min_{\partial\Omega_h} \phi_h \\
&sgn(\nabla\phi_h) \geq 0 \text{ or } sgn(\nabla\phi_h) \leq 0
\end{aligned}$$

Compared to the general model above, we will consider pure Dirichlet boundary value problem for simplicity. Besides, we will only consider the 2D problem. Note that this optimization will have both equality and inequality constraints, partial differential equation and algebraic equation constraints. The PDE itself is linear and we will consider only the constant advective velocity $\mathbf{b}$, diffusivity ν and reaction parameter c. In addition, the positivity, discrete maximum principle and monotonicity are to be implemented as last three constraints above, respectively.

4 Mathematical analysis

Write out the finite difference collocation schemes for the ADR system and apply constrained optimization theory to it.

[1] `https://github.com/Grayson3455/ADR-pyomo.dae`

5 Numerical examples

5.1 Smooth problem

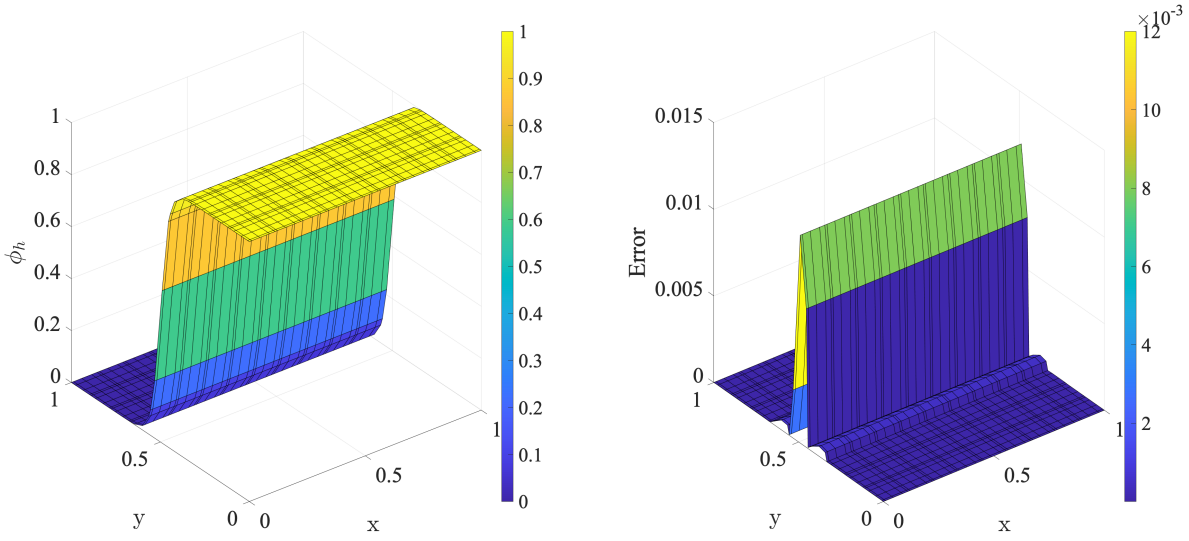
We first test the functionality of `pyomo.dae` by solving advection-diffusion-reaction equations with known solutions. This is often referred as the test by manufactured solutions, *i.e.*, we design a solution profile $\phi(x, y)$ and then plug it to the PDE to solve for the source/sink term f. The example we followed was originally proposed by [3], which is characterized by a hyperbolic tangent distribution only in $x-$direction:

$$\phi(x, y) = \frac{1}{2}(1 - \tanh(\frac{x - 0.5}{\lambda}))$$

With the parameters $\mathbf{b} = (1, 0)$, $\nu = 10^{-5}, c = 1$, $\lambda = 0.05$. Note that the magnitude of λ controls the slope of the hyperbolic function, which brings an interior layer near $x = 0.5$.

5.1.1 Case1: Steady advection-diffusion-reaction equation

Solved by the finite difference collocation solver provided in `pyomo.dae`, we have the following results:



(a) Solution profile

(b) Nodal error distribution

Figure 1: `pyomo` solution of the manufactured advection-diffusion-reaction problem

Note that with only 9 finite elements and 3 collocation points each, we still have a decent approximation. Also, it's clear the error peaks near the interior layer, where sharp solution gradient brings less accuracy by polynomial-based numerical approximations.

5.1.2 Case2: Unsteady advection-diffusion-reaction equation

Without creating the time-dependent part, we initiated a parabolic function as the initial condition and start time-marching. Theoretically, the steady state will be reached by a few time steps.

5.2 Non-smooth problem

So far, we did not see any spurious oscillations, over/undershoots triggered by numerical instabilities. Besides the robust numerical property provided by the collocation scheme, the smoothness assumption of the problem also makes a contribution. In this section, we will consider the so-called skewed advection problem, where the special structure of the boundary conditions brings discontinuity and immediately creates a singularity-perturbed problem.

5.2.1 Case1: Steady advection-diffusion-reaction equation

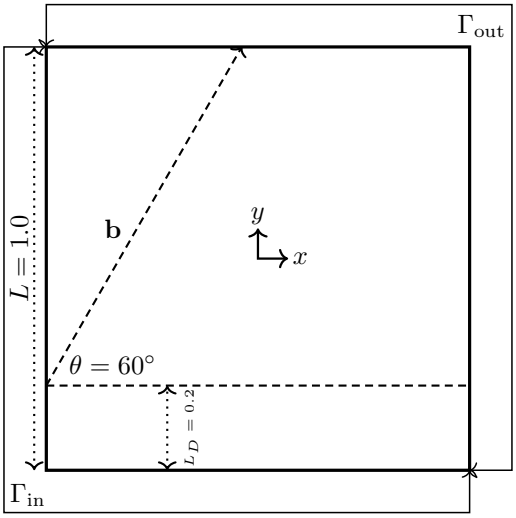


Figure 2: Sketch of the skewed advection model

5.2.2 Case2: Unsteady advection-diffusion-reaction equation

6 Conclusions

References

[1] Lorenz T Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.

[2] Erik Burman and Alexandre Ern. Stabilized galerkin approximation of convection-diffusion-reaction equations: discrete maximum principle and convergence. *Mathematics of computation*, 74(252):1637–1652, 2005.

[3] Erik Burman and Peter Hansbo. Edge stabilization for galerkin approximations of convection–diffusion–reaction problems. *Computer methods in applied mechanics and engineering*, 193(15-16):1437–1453, 2004.

[4] Zhenhua Chai and TS Zhao. Lattice boltzmann model for the convection-diffusion equation. *Physical Review E*, 87(6):063309, 2013.

[5] John A Evans, Thomas JR Hughes, and Giancarlo Sangalli. Enforcement of constraints and maximum principles in the variational multiscale method. *Computer Methods in Applied Mechanics and Engineering*, 199(1-4):61–76, 2009.

[6] Volker John, Petr Knobloch, and Julia Novo. Finite elements for scalar convection-dominated equations and incompressible flow problems: a never ending story? *Computing and Visualization in Science*, 19(5-6):47–63, 2018.

[7] RD Lazarov, Ilya D Mishev, and Panayot S Vassilevski. Finite volume methods for convection-diffusion problems. *SIAM Journal on Numerical Analysis*, 33(1):31–55, 1996.

[8] Bethany Nicholson, John D Siirola, Jean-Paul Watson, Victor M Zavala, and Lorenz T Biegler. pyomo. dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations. *Mathematical Programming Computation*, 10(2):187–223, 2018.

[9] Martin Stynes and David Stynes. *Convection-Diffusion Problems : An Introduction to Their Analysis and Numerical Solution*. American Mathematical Society, 2018.

Appendix 1: Computer code(steady ADR example)

```
from pyomo.environ import *
from pyomo.dae import *
import numpy as np
from matplotlib import pyplot as plt

# space-time variables
m      = ConcreteModel()
m.x    = ContinuousSet(bounds = (0,1))
m.y    = ContinuousSet(bounds = (0,1))
m.phi  = Var(m.x, m.y)

# define derivative variables
m.DphiDx   = DerivativeVar(m.phi, wrt = m.x)
m.DphiDy   = DerivativeVar(m.phi, wrt = m.y)
m.DphiDx_2 = DerivativeVar(m.phi, wrt = (m.x, m.x))
m.DphiDy_2 = DerivativeVar(m.phi, wrt = (m.y, m.y))

# define PDE parameters
m.c     = Param(initialize = 1.0)  # the reaction constant
m.nu    = Param(initialize = 1e-5) # the diffusivity
m.bx    = Param(initialize = 1.0)  # the advection velocity, x-component
m.by    = Param(initialize = 0.0)  # the advection velocity, y-component
m.lmd     = Param(initialize = 0.05) # slope of manufactured solution

# create lambda function for manufactured solutions
u_e     = lambda x : 0.5* ( 1.0 - np.tanh((x - 0.5)/value(m.lmd)) )
du_e    = lambda x : -1.0/(2.0*value(m.lmd)) \
                  * (1.0/np.cosh((x-0.5)/value(m.lmd)))**2
du_e2   = lambda x : 1.0/value(m.lmd)/value(m.lmd) * \
                  (1.0/np.cosh((x-0.5)/value(m.lmd)))**2 * np.tanh((x - 0.5)/value(m.lmd))
f       = lambda x,y : du_e(x) - value(m.nu) * du_e2(x) + value(m.c)*u_e(x)

# define adr pde, i: x, j: y
def ADR_pde(m,i,j):
        if i == 0 or i  == 1: # boundary conditions to be implemented, no constraints
                return Constraint.Skip
        return m.bx * m.DphiDx[i,j] + m.by * m.DphiDy[i,j]- m.nu*m.DphiDx_2[i,j] \
        - m.nu*m.DphiDy_2[i,j]  + m.c*m.phi[i,j] == f(i,j)
m.pde = Constraint(m.x, m.y, rule = ADR_pde )

# left boundary condition [phi = 1 @ x = 0]
def BC1(m,j):
        return m.phi[0,j] == 1.0
m.BCx_0 = Constraint(m.y, rule = BC1)

# right boundary condition [phi = 0 @ x = 1]
def BC2(m,j):
        return m.phi[1,j] == 0.0
m.BCx_1 = Constraint(m.y, rule = BC2)

# trivial obj
m.obj = Objective(expr = 1)
```

```
# discretization and solve
discretizer = TransformationFactory('dae.collocation')
discretizer.apply_to(m,nfe=9,ncp=4,wrt=m.x)
discretizer.apply_to(m,nfe=9,ncp=4,wrt=m.y)

solver = SolverFactory('ipopt')
results = solver.solve(m, tee=True)
```