ICMIEE20-080

A Systematic Approach to Solve Unsteady Thermodynamic Problems using Python and Its Open-source Packages

Monjur Morshed[*], *Tahir Mahmud*
Department of Energy Science and Engineering, Khulna University of Engineering & Technology, Khulna-9203, BANGLADESH

ABSTRACT
Although thermodynamics is used for steady state design and performance evaluation of thermal systems, unsteady problems are important from system dynamics and control point of view. From dynamics perspective it is desirable to predict the system's pressure, temperature, and mass as a function of time. However, the governing conservation equations give rise to system of nonlinear coupled ordinary differential equations involving mass and energy, and solution of this system requires numerical methods. There exists a lack of textbooks that deals with unsteady thermodynamic problems except some that use proprietary software packages. Furthermore, in academia and industry state-of-the-art thermodynamic simulation are carried out using Engineering Equation Solver (EES), REFPROP, FLUID-PROP, Cycle-Tempo, Aspen HYSYS, MATLAB or similar software. However, the programming language Python and it open-source packages offer an excellent ecosystem that can be readily adopted to address thermodynamic simulation problems that is comparable to proprietary software. The challenge being though there exits little literature that discusses systematic solution strategy using Python. This paper presents a systematic solution approach to unsteady thermodynamic problems using Python and its open-source packages.

Keywords: Unsteady thermodynamics, ODE systems, numerical solution, Python

1. Introduction

Engineering thermodynamics along with its laws plays the most important role of uniting heat transfer and fluid mechanics when it comes to design, simulate, optimize, and performance evaluation of thermal and energy conversion systems. While much of the attention is given what is known as steady state simulation, dynamical behavior of thermodynamic systems is important from operation and control perspective as it is desirable to predict the system's pressure, temperature, mass, and energy as a function of time. The governing conservation equations of thermodynamics give rise to a system of nonlinear coupled ordinary differential equations involving mass and energy, and solution of this system requires numerical methods. As thermodynamics is taught mostly during sophomore year in engineering curricula, text books [1-4] tend to focus more on steady state analysis rather than unsteady thermodynamic problems as students are not trained in numerical methods until junior year. There are some books [5-7] that treat unsteady thermodynamics but use proprietary software that are expensive and require training. Moreover, in academia and industry state-of-the-art thermodynamic simulations are carried out using Engineering Equation Solver (EES) [8], REFPROP [9], FLUID-PROP [10], Cycle-Tempo [11], Aspen HYSYS [12], MATLAB [13], or similar software. Hence, a wealth of engineering knowledge remains untouched due to expensive proprietary software and lack of literature that discuss the know-how regarding unsteady thermodynamic simulation using open-source software and packages. The programming language Python [14] and its open-source packages offer an excellent ecosystem that can be readily adopted to address thermodynamic simulation problems that is comparable to proprietary software [15]. This paper discusses a systematic solution approach to unsteady thermodynamic problems using Python and its open-source packages while demonstrating the technique by applying it to a particular example problem.

2. Theoretical Background

In order to tackle unsteady thermodynamic problems, conservation equations of mass and energy as well as equation of state of working fluid are needed. As thermodynamics deals with equilibrium states, there can be no spatial distribution of the variables and lumped parameter model has to be adopted together with one-dimensional flow assumptions [2]. For an open system, the rate equations for conservation of mass and energy are as follows.

$$\frac{dm_{cv}}{dt} = \sum_i \dot{m}_i - \sum_e \dot{m}_e \tag{1}$$

$$\frac{dE_{cv}}{dt} = \dot{Q}_{cv} - \dot{W}_{cv} + \sum_i \dot{m}_i \Phi_i - \sum_e \dot{m}_e \Phi_e \tag{2}$$

where,

$$\Phi_i = \left(h_i + \frac{V_i^2}{2} + g z_i \right)$$

$$\Phi_e = \left(h_e + \frac{V_e^2}{2} + g z_e \right)$$

[*] Corresponding author. Tel.: +88-01741219395
E-mail addresses: monjur@ese.kuet.ac.bd

The equation of state of a working fluid can be expressed in an implicit form as

$$F(p,T,v)=0 \tag{3}$$

Eq. (1) & (2) are ordinary differential equations where time is the independent variable and mass (m_{cv}) and energy (E_{cv}) of the control volume are dependent variables respectively. E_{cv} denotes the total energy of the control volume also, which is the summation of internal energy, kinetic energy, and potential energy, and it is extensive in nature. If the problem in hand possesses stationary control volume then change is kinetic and potential energy is zero leaving internal energy as representative of the total energy of the system. Unlike Eq. (1) & (2), Eq. (3) is a nonlinear equation involving pressure (p), temperature (T), and specific volume (v). Some of the physical quantities appearing on the right sides of Eq. (1) and (2) generally depend on the system's properties such as pressure and temperature. Short descriptions of these important quantities are as follows.

2.1 Mass flow rate, $\dot{m}$

With one-dimensional flow assumptions, the mass flow rate of a fluid entering or leaving an open system can be calculated with the following equation if the average flow velocity (V) and flow area (A) are known.

$$\dot{m}=\rho VA \tag{4}$$

The density of fluid (ρ) is a function of pressure and temperature, thus can be evaluated using Eq. (3). In practical settings where valves are used to regulate fluid flow, the mass flow rate can be calculated by empirical correlations involving pressure difference and density provided by the valve theory or valve manufacturer.

2.2 Heat transfer rate, $\dot{Q}_{cv}$

Heat transfer to or from an open system can be modeled by three different models – conduction, convection, radiation, depending on the physics involved. Equation for these modes of heat transfer can be found in [16, 17].

2.3 Work transfer rate, $\dot{W}_{cv}$

There can be different types of work transfer rate involved in a thermodynamics problem such as expansion or compression of a fluid, stretching a thin film, moving a force through a linear distance, rotating shaft work, electric work etc. Equation for these work transfer modes can be found in [2, 5].

2.4 Specific enthalpy, h

Specific enthalpy is an intensive property that is defined as the summation of internal energy and product of pressure, and specific volume of the fluid in consideration. The value of specific enthalpy can be evaluated as a function of pressure and temperature.

3. Necessary Python Packages and Modules

Before describing the systematic solution strategy and its implementation using Python, it is necessary to first observe the mathematical nature of Eq. (1) & (2) and brief introduction to Python packages and modules. Each of the equation is first order non-linear ordinary differential equations (ODE), and together they constitute a system of first order non-linear coupled ODE system. Furthermore, they can be expressed in canonical vector form in following way.

$$\frac{d}{dt}\begin{bmatrix} m_{cv} \\ E_{cv} \end{bmatrix}=\begin{bmatrix} \sum_i \dot{m}_i-\sum_e \dot{m}_e \\ \dot{Q}_{cv}-\dot{W}_{cv}+\sum_i \dot{m}_i\Phi_i-\sum_e \dot{m}_e\Phi_e \end{bmatrix} \tag{5}$$

Or,

$$\frac{d}{dt}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}=\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{6}$$

In Eq. (6), y_1 and y_2 represent m_{cv} and E_{cv} , and f_1 and f_2 represent right side of Eq. (1) and (2) respectively. Representing a system of ODE in canonical form is necessary because Python function `odeint`, which resides in `scipy.integrate` [18] module, requires the system in this form in order to perform numerical integration. In order to give Python matrix computation capabilities like MATLAB, the `numpy` [19] package is used.

Table 1 List of necessary Python libraries.

Library or package	Usage
`numpy`	Array and matrix handling
`scipy.integrate`	Numerically solve ODEs
`matplotlib`	Plotting and visualization
`CoolProp`	Thermodynamic property

As mentioned earlier that some quantities on the right side of the equations depend on pressure and temperature, solution of Eq. (5) needs to be used to calculate back pressure and temperature based on state postulate, which states that two independent intensive properties are necessary to fix other thermodynamic state properties. The package `CoolProp` [20] contains the function `PropsSI` that can be used to access thermodynamic property of fluids in this regard. This package can be thought of open-source replacement of REFPROP. Finally, in order to visualize the simulated results Python package `matplotlib` [21] is used. Table 1 provides the list of Python library and modules necessary to implement the simulation.

4. Systematic Solution Strategy

In order to drive the systematic solution strategy home, an example of unsteady thermodynamic problem from [5] will be used. A flow chart of the computer program which delineates the systematic solution strategy is shown in Fig.1, and description of the strategy will follow the steps mentioned in that chart.

The problem deals with filling of a hydrogen storage tank of a vehicle that is use for fuel cell. Hydrogen is supplied from a fuel reservoir at constant pressure and temperature to fill three heavy-walled steel storage cylinders. The cylinders are connected to the fuel reservoir via a valve. The problem seeks the solution of predicting the pressure, temperature of the cylinders, and amount of hydrogen stored during the filling process as a function of time.

The problem provides two auxiliary equations to predict mass flow rate through the valve and convective heat transfer from the cylinders, and they are

$$\dot{m} = C_{valve}\sqrt{p_{supply} - p(t)} \tag{7}$$

$$\dot{Q} = h_{conv} A_s [T_{wall} - T(t)] \tag{8}$$

Table 2 lists all the parameters provided in the problem with their respective numeric values and unit.

Table 2 List of parameters and their values.

Parameters	Value	Unit
Number of tanks, N	3	-
Height of tanks, H	65	cm
Inner diameter of tanks, d	28	cm
Ambient temperature, T_{amb}	25	ºC
Wall temperature, T_{wall}	25	ºC
Supply pressure, p_{supply}	300	bar
Supply temperature, T_{supply}	25	ºC
Valve coefficient, C_{valve}	2.68×10^{-6}	kg s^{-1} $Pa^{-0.5}$
Conv. coefficient, h_{conv}	40	W m^{-2} K^{-1}

As the hydrogen storage cylinders are stationary, the change in total energy of the system is represented by the change in internal energy only. The cylinders have one incoming mass flow, and the flow rate is given by Eq. (7), where it depends on instantaneous system pressure that must be calculated once the solution of ODEs is found. Furthermore, the convective heat transfer depends on instantaneous system temperature, which again must be calculated from solution of ODEs. Other assumptions are that the kinetic and potential energy of the incoming hydrogen are negligible compared to its enthalpy. With these observations, the governing system of ODEs for the problem takes the following form.

$$\frac{d}{dt}\begin{bmatrix} m_{cv} \\ U_{cv} \end{bmatrix} = \begin{bmatrix} \dot{m} \\ \dot{Q} + \dot{m}h_{sply} \end{bmatrix} \tag{9}$$

Where, $\dot{m}$ and $\dot{Q}$ are given by Eq. (7) and (8) respectively.

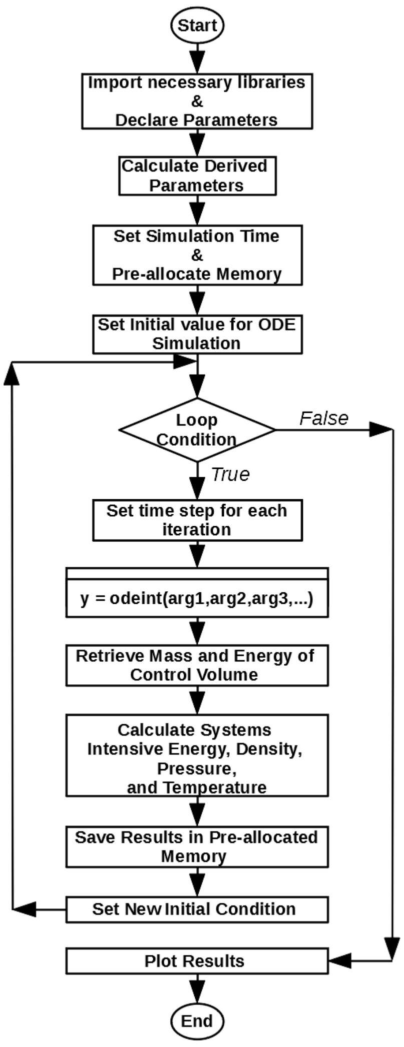
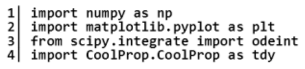

Start
Import necessary libraries & Declare Parameters
Calculate Derived Parameters
Set Simulation Time & Pre-allocate Memory
Set Initial value for ODE Simulation
Loop Condition
False
True
Set time step for each iteration
y = odeint(arg1,arg2,arg3,...)
Retrieve Mass and Energy of Control Volume
Calculate Systems Intensive Energy, Density, Pressure, and Temperature
Save Results in Pre-allocated Memory
Set New Initial Condition
Plot Results
End

Fig.1 Flow chart of the computer program.

4.1 Importing packages

The first step to systematically solve unsteady thermodynamic problem using Python is to import necessary packages, modules, and functions. Fig.2 lists the code necessary for this step.

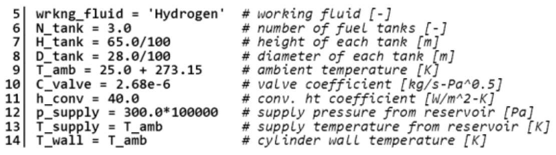


```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import CoolProp.CoolProp as tdy
```

Fig.2 Python code for importing packages and modules.

4.2 Declare parameters

The next step is to assign parameters that come from the geometry, fluid mechanics, heat transfer, and thermodynamics. These quantities remain constant all the time for a particular simulation. Fig.3 lists the code used to declaring the parameters for this particular problem as listed in Table 2.

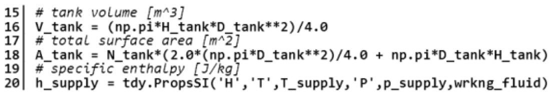


```python
wrkng_fluid = 'Hydrogen'  # working fluid [-]
N_tank = 3.0              # number of fuel tanks [-]
H_tank = 65.0/100         # height of each tank [m]
D_tank = 28.0/100         # diameter of each tank [m]
T_amb = 25.0 + 273.15     # ambient temperature [K]
C_valve = 2.68e-6         # valve coefficient [kg/s-Pa^0.5]
h_conv = 40.0             # conv. ht coefficient [W/m^2-K]
p_supply = 300.0*100000   # supply pressure from reservoir [Pa]
T_supply = T_amb          # supply temperature from reservoir [K]
T_wall = T_amb            # cylinder wall temperature [K]
```

Fig.3 Python code for parameter declaration.

4.3 Calculate derived parameters

Generally there are quantities that are derived from the parameter of a problem. For example, the problem in hand tank volume, tank surface area, and specific enthalpy of supplied hydrogen are derived parameters. Fig.4 lists the code for derived parameters.

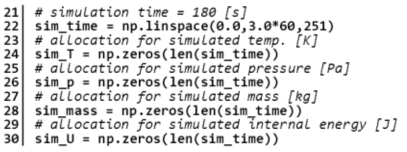


```python
# tank volume [m^3]
V_tank = (np.pi*H_tank*D_tank**2)/4.0
# total surface area [m^2]
A_tank = N_tank*(2.0*(np.pi*D_tank**2)/4.0 + np.pi*D_tank*H_tank)
# specific enthalpy [J/kg]
h_supply = tdy.PropsSI('H','T',T_supply,'P',p_supply,wrkng_fluid)
```

Fig.4 Python code for derived parameters.

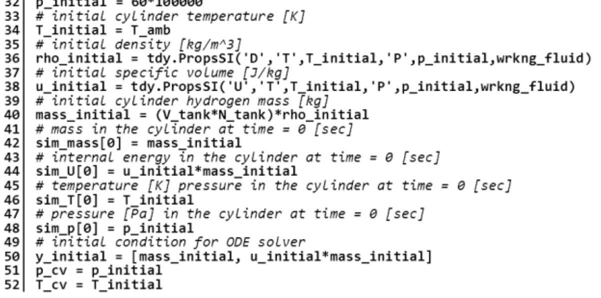
4.4 Set simulation time and memory pre-allocation

In this step the total simulation time has to be setup. For this purpose `linspace` function from `numpy` can be used that allows specifying the number of grid points one wishes to have at which the numerical solution of ODEs will be carried out by `odeint` function. Memory pre-allocation is another important task in this step. It allows the programmer to come up with predefined variables that are of same array size of total simulation time so that during the simulation these variables can store the results. Memory pre-allocation helps the computer code to run faster. Fig.5 lists the necessary code used in this step for the particular problem in hand.



```python
# simulation time = 180 [s]
sim_time = np.linspace(0.0,3.0*60,251)
# allocation for simulated temp. [K]
sim_T = np.zeros(len(sim_time))
# allocation for simulated pressure [Pa]
sim_p = np.zeros(len(sim_time))
# allocation for simulated mass [kg]
sim_mass = np.zeros(len(sim_time))
# allocation for simulated internal energy [J]
sim_U = np.zeros(len(sim_time))
```

Fig.5 Python code for memory pre-allocation.



```python
# initial cylinder pressure [Pa]
p_initial = 60*100000
# initial cylinder temperature [K]
T_initial = T_amb
# initial density [kg/m^3]
rho_initial = tdy.PropsSI('D','T',T_initial,'P',p_initial,wrkng_fluid)
# initial specific volume [J/kg]
u_initial = tdy.PropsSI('U','T',T_initial,'P',p_initial,wrkng_fluid)
# initial cylinder hydrogen mass [kg]
mass_initial = (V_tank*N_tank)*rho_initial
# mass in the cylinder at time = 0 [sec]
sim_mass[0] = mass_initial
# internal energy in the cylinder at time = 0 [sec]
sim_U[0] = u_initial*mass_initial
# temperature [K] pressure in the cylinder at time = 0 [sec]
sim_T[0] = T_initial
# pressure [Pa] in the cylinder at time = 0 [sec]
sim_p[0] = p_initial
# initial condition for ODE solver
y_initial = [mass_initial, u_initial*mass_initial]
p_cv = p_initial
T_cv = T_initial
```

Fig.6 Python code for setting up initial conditions.

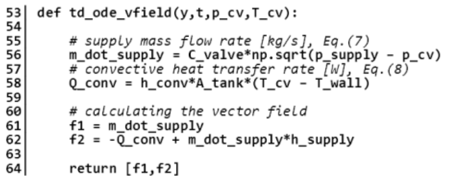
4.5 Set initial values

For solving system of ODEs, initial conditions needs to be set up. The initial pressure and temperature of the system should be used calculate density of the system which will be used to calculate system's initial internal energy and mass. Furthermore, these values should be assigned to the first entry of respective arrays that have been created in the earlier memory pre-allocation step. Finally initial condition for the ODE solver is set up for first iteration step. Fig.6 shows the relevant Python code.

4.6 ODE solution loop

Among all the steps this is most crucial one. The basic idea of the loop is that each iteration solves the ODE system, saves the results in pre-allocated variables, and sets up a new initial value problem for the next iteration to solve. The loop runs for $n-1$ times where n is the number of grid points in the total simulation time.

The heart of solution loop is the use of `odeint` function [22]. This function takes three mandatory inputs, namely, a user defined function that returns the vector field of an ODE system, initial condition, and time step for which the ODE system needs to be solved. Additional parameters can be passed into the function as a tuple. The user defined function, which is the first argument of the `odeint` function, has certain structure too. It must have two mandatory inputs, namely, a vector containing the value of unknown functions of ODEs and time. Additional parameters should come after these two inputs. Regarding unsteady thermodynamic problems, these additional parameters are pressure and temperature of the system. This function should calculate the f_1 and f_2 terms and return a list containing these values. Fig.7 shows the code for defining the user defined function for the example problem.



```python
def td_ode_vfield(y,t,p_cv,T_cv):

    # supply mass flow rate [kg/s], Eq.(7)
    m_dot_supply = C_valve*np.sqrt(p_supply - p_cv)
    # convective heat transfer rate [W], Eq.(8)
    Q_conv = h_conv*A_tank*(T_cv - T_wall)

    # calculating the vector field
    f1 = m_dot_supply
    f2 = -Q_conv + m_dot_supply*h_supply

    return [f1,f2]
```

Fig.7.Python code of the function defining vector field of the ODE system.

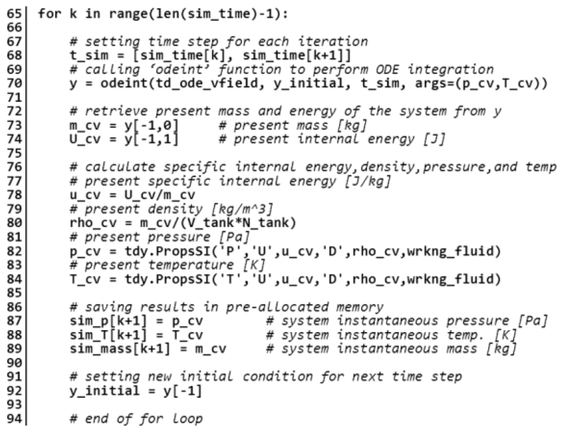
The `odeint` function returns an array containing solution functions, namely, mass of the system and energy, when used to solve Eq. (5). As solution of system if ODEs provides mass and energy of the system, they should be used to calculate specific internal energy and specific volume according to Eq. (10).

$$u = \frac{U_{cv}}{m_{cv}} \text{ and } \rho = \frac{m_{cv}}{Volume} \tag{10}$$

The reason being these two quantities will be used to calculate system pressure and temperature using

`PropsSI` function [23] provided by the `CoolProp` library.

The next steps involve saving the results found in the current iteration and set new initial condition for next loop execution. Fig.8 lists all the python code used in the ODE solution loop for the particular problem in hand.



```python
for k in range(len(sim_time)-1):

    # setting time step for each iteration
    t_sim = [sim_time[k], sim_time[k+1]]
    # calling ‘odeint’ function to perform ODE integration
    y = odeint(td_ode_vfield, y_initial, t_sim, args=(p_cv,T_cv))

    # retrieve present mass and energy of the system from y
    m_cv = y[-1,0]     # present mass [kg]
    U_cv = y[-1,1]     # present internal energy [J]

    # calculate specific internal energy,density,pressure,and temp
    # present specific internal energy [J/kg]
    u_cv = U_cv/m_cv
    # present density [kg/m^3]
    rho_cv = m_cv/(V_tank*N_tank)
    # present pressure [Pa]
    p_cv = tdy.PropsSI('P','U',u_cv,'D',rho_cv,wrkng_fluid)
    # present temperature [K]
    T_cv = tdy.PropsSI('T','U',u_cv,'D',rho_cv,wrkng_fluid)

    # saving results in pre-allocated memory
    sim_p[k+1] = p_cv        # system instantaneous pressure [Pa]
    sim_T[k+1] = T_cv        # system instantaneous temp. [K]
    sim_mass[k+1] = m_cv     # system instantaneous mass [kg]

    # setting new initial condition for next time step
    y_initial = y[-1]

    # end of for loop
```

Fig.8 Python code for ODE solution loop.

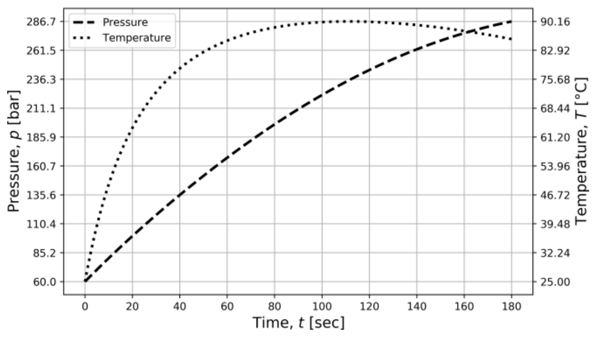


Fig.9 Pressure and temperature as function of time.

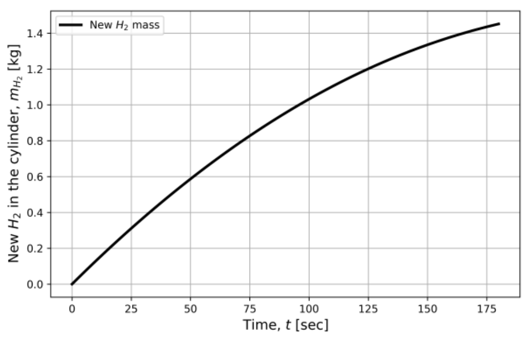


Fig.10 Hydrogen introduced to storage tank as function of time.

Once the simulation is complete inside the loop, the stored results can be used to plot the results of interest i.e. system’s pressure, temperature, mass as functions of time using Python’s `matplotlib` library. Fig.9 and 10 show the results respectively.

5. Discussion

In this paper a systematic solution approach of nonlinear coupled ordinary differential equations involving mass and energy that arise in unsteady thermodynamic problems is presented by using python and its open-source packages. In case of simulating an unsteady thermal engineering problem, the traditional approach has been to convert energy rate balance equation into temperature or pressure rate equation with the help of basic thermodynamic relations and equation of state of the working substance involved and resort to numerical techniques like Ranga-Kutta method. The use of thermodynamic relations makes the derivation of the rate equations tedious and complex to handle when coding the solution technique using a programming language. Furthermore, in order to avoid complexity of dealing with real fluid, a simplifying assumption like ideal gas model is sometimes adopted, which may hampers the accuracy and applicability of the simulation results.

6. Conclusion

The prescribed method delineated in this paper has several advantages over the traditional approach. First, this method allows simulating unsteady thermodynamic problems using mass and energy rate balance equations rather than converting the energy rate balance equation to temperature or pressure rate equations. Such advantage is gained by using the Gibbs phase rule and state postulate, which allow evaluation of pressure and temperature or other intensive properties of the fluid involved with thermo-physical property library like `CoolProp`. Secondly, this method allows us to arrange the governing equations in canonical ODE form that can be readily solved by using high precision numerical ODE integration schemes available in `Scipy` library. Moreover, if there are subsystems present in the problem that is also governed by ODEs, then arranging those equations along with the thermodynamic ODEs in a canonical form enhances the modularity and reduces the simulation complexity. Third, as thermo-physical property library is used, there is no need to make simplified assumptions like ideal gas. Finally, if time varying control volume is involved, it can also be dealt with by deriving rate of change of volume from the system’s geometry.

7. References

[1] Çengel, Y., Boles, M. A., Thermodynamics, McGraw-Hill Education, Ed. 9, New York, 2019.

[2] Moran, M. J., Shapiro, H. N., Boettner, D. D., Bailey, M. B., Fundamentals of Engineering Thermodynamics, Wiley, Ed. 9, New York, 2018.

[3] Borgnakke, C., Sonntag, R. E., Fundamentals of thermodynamics, Wiley, Ed. 8, New York, 2013.

[4] Balmer, R. T., Modern Engineering

Thermodynamics, Academic Press, Ed. 1, New York, 2011.
[5] Klein, S., Nellis, G., Thermodynamics, Cambridge University Press, Ed. 1, New York, 2012.
[6] Reynolds, W. C., Colonna, P., Thermodynamics, Cambridge University Press, Ed. 1, Cambridge, 2018.
[7] Massoud, M., Engineering Thermofluids, Springer, Ed. 1, Berlin, 2005.
[8] http://www.fchartsoftware.com/ees/ (10-08-2020)
[9] https://www.nist.gov/srd/refprop (10-08-2020)
[10] http://www.asimptote.nl/software/fluidprop (10-08-2020)
[11] http://www.asimptote.nl/software/cycle-tempo/ (10-08-2020)
[12] https://www.aspentech.com/en/products/engineering/aspen-hysys (10-08-2020)
[13] https://www.mathworks.com/products/matlab.html (10-08-2020)
[14] https://www.python.org (10-08-2020)
[15] Zoder, M., Balke, J., Hofmann, M., Tsatsaronis, G., Simulation and Exergy Analysis of Energy Conversion Processes Using a Free and Open-Source Framework—Python-Based Object-Oriented Programming for Gas- and Steam Turbine Cycles. *Energies,* vol. 11(10):2609, 2018.
[16] Incropera, F. P., DeWitt, D. P., Bergman, T. L., Lavine, A.S., Fundamentals of Heat and Mass Transfer, Wiley, Ed. 8, New York, 2017.
[17] Nellis, G., Klein, S., Heat Transfer, Cambridge University Press, Ed. 1, New York, 2012.
[18] https://docs.scipy.org/doc/scipy1.5.2/reference/integrate.html (10-08-2020)
[19] https://numpy.org (10-08-2020)
[20] http://www.coolprop.org/index.html (10-08-2020)
[21] https://www.matplotlib.org (10-08-2020)
[22] https://docs.scipy.org/doc/scipy1.5.2/reference/generated/scipy.integrate.odeint.html#scipy.integrate.odeint (10-08-2020)
[23] http://www.coolprop.org/coolprop/HighLevelAPI.html (10-08-2020)

NOMENCLATURE

m	: Mass, kg
$\dot{m}$	: Mass flow rate, kg·s^{-1}
E	: Energy, J
h	: Specific enthalpy, J·kg^{-1}
p	: Pressure, Pa or bar
T	: Temperature, K or °C
A_s	: Flow area, m^2
A	: Cylinder surface area, m^2
V	: Average velocity, m·s^{-1}
$\dot{Q}$	: Heat transfer rate, J·s^{-1}
$\dot{W}$	: Work transfer rate, J·s^{-1}