Brown University

ENGN 2910Q: Professor Goldsmith

1D and 2D Flow with Catalytic Surface Chemistry

Kento Abeywardane

April 14, 2021

Table of Contents

Introduction……..... **1**

Background……...……...……**1**

Method and Results…..…….……..**4**

1D PFR with Heterogeneous Chemistry ……...**4**

2D Reactor with Heterogeneous Chemistry ……..**8**

Conclusion and Future Considerations ……...……....…….**12**

References…….**13**

Appendices…….…….…..**14**

Symbols ……**14**

Appendix Figures……...**15**

Appendix Code……..**16**

Introduction

This project aims to create a 2-dimensional, steady state model of an H_2-O_2 mixture flowing through a rectangular reactor with a Platinum catalytic surface. Transport of species moving through the reactor and chemistry occurring heterogeneously must be accurately modeled. The system's governing equations are described by a system of differential algebraic equations (DAE). Due to the mathematically complex system, this problem was computationally solved with Python using the Method of Lines. The corresponding packages utilized were Numpy and Scikits.odes.dae for mathematical operations and Cantera for chemical properties and kinetics. This report outlines the methods of creating a model, first as a 1D PFR, and finally a 2D reactor.

Background

Chemically reacting flow can be broken down into two broad categories of fluid flow and chemistry, as the name suggests. In multi-component fluid flow without reaction, there are five main governing equations. These are the conservation of mass, momentum (Navier-Stokes), Energy, N-1 Species, and the equation of state. Coupled together, they can accurately describe laminar flow, keeping track of variables such as density, temperature, pressure, velocities, and species fractions. When homogenous reactions are involved, net production rates of species from within the fluid (denoted $\dot{\omega}_k$) must be incorporated into the conservation of energy and conservation of species equations. Further, the addition of heterogeneous reactions alter the conservation of mass, energy, and species at the surface boundary by adding net production rates (denoted $\dot{s}_k$) of gas phase species from the surface.

The additions of chemistry to the already complex transport equations make analytical solutions impossible and complicates computational methods. It becomes necessary to apply simplifications in the dimensionality of the problem and/or implement certain assumptions. For this project, the goal was to simulate a chemically reacting flow at steady state and so the governing equations were simplified to be as follows[2][4]:

Conservation of Mass:

$$\nabla \cdot (\rho \vec{v}) = \frac{A}{V} \sum^{N_{gas}} \dot{s}_k W_k \qquad \textbf{Eq. 1}$$

Conservation of Momentum:

$$\rho(\vec{v} \cdot \nabla)\vec{v} = \rho\vec{g} - \nabla p + \nabla(\mu(\nabla \cdot \vec{v} + (\nabla \cdot \vec{v})^T) + K(\nabla \cdot \vec{v})\vec{T}) \qquad \textbf{Eq. 2}$$

Conservation of Energy (ideal gas):

$$\rho\overline{c_p}\vec{v} \cdot \nabla T = \nabla \cdot (\lambda \nabla T) - \sum^{N_{gas}} c_{p,k}\vec{j_k} - \sum^{N_g as} h_k \dot{\omega_k} W_k \qquad \textbf{Eq. 3}$$

Conservation of Species (N–1):

$$\rho\vec{v} \cdot \nabla y_k = -\nabla \cdot \vec{j_k} + \frac{A}{V}\dot{s_k}W_k - y_k\frac{A}{V}\sum^{N_{gas}} \dot{s}_k W_k + \dot{\omega_k}W_k, \qquad k = 1...N_g \qquad \textbf{Eq. 4}$$

Equation of State

$$\rho = f(T, p) \qquad \textbf{Eq. 5}$$

Surface Species Production

$$\dot{s_k} = 0, \qquad k = 1...N_s \qquad \textbf{Eq. 6}$$

Gas and Surface Species Fractions

$$\sum^{N_{gas}} y_k = 1 \qquad \textbf{Eq. 7}$$

$$\sum^{N_{surf}} Z_k = 1 \qquad \textbf{Eq. 8}$$

Eqs. 1-5 are the partial differential equations while **Eq. 6-8** are the algebraic constraints. The surface reaction terms are only true for the boundary layer at the catalytic surface and it should be noted that **Eq. 6** was simplified from a differential equation to an algebraic constraint because the transient part of $\frac{dZ_k}{dt} = \dot{s_k}W_k$ vanishes at steady state. This means that the net production rate of surface phase species remains constant. **Eqs. 7** and **8** are constraints that ensure that the gas phase species and surface site fractions sum to 1.

The chemistry that is investigated in this project is H_2/O_2 combustion with platinum as a heterogeneous catalyst. All necessary chemistry and thermodynamic properties are derived from

the Python library Cantera[N2] (v. 2.5.0). Specifically, the *'ptcombust.yaml'* file (which utilizes mechanisms from *'gri30.yaml'*) is used.

As noted before, this system of equations is known as Differential Algebraic Equations (DAE). The addition of algebraic equations makes the computation significantly harder for a numerical solver so normal ODE integrators cannot be used. For this project, the open source package of Python's Scikit-odes's DAE[N3] interface (scikit.odes.dae) was utilized to solve the simulations. The specific solver used was IDA developed and maintained by the Lawrence Livermore National Laboratory. This solver works by inputting an initial array of variables and derivatives of those variables at an initial time (or position given the dimension integrated along). The governing equations are placed within the function as residuals. The solver automatically adjusts its time/position steps as it solves the equations and outputs an array of values for the variables at each iteration.

A 2D system of PDEs, with or without algebraic constraints, can be solved by using the Method of Lines (MoL). The MoL replaces derivatives in one dimension with algebraic approximations such that the system of PDEs is simplified to be a system of ODEs[5]. Thus, it is better said that the system is discretized in one dimension and integrated along the other. For the 2D reactor, MoL is directly used and the respective governing equations of those will be explored later in this paper.

A plug flow reactor (PFR) is a duct with a steady flow of reacting fluid. It is assumed that the flow inside the reactor is turbulent such that the fluid is perfectly mixed in the radial direction. At steady state, and the assumption that there is no dispersion in the axial direction, the PFR is automatically simplified to a 1D reactor. Initially solving this lower dimensional reactor will aid in the process of modeling a 2D reactor since the same overall method applies, but with the extra step of discretization in the second dimension for MoL.

Method and Results

1D PFR with Heterogeneous Chemistry

First, the governing equations for the case of an adiabatic 1D PFR with homogeneous and heterogeneous chemistry was formulated and modeled using the DAE solver in Python. Note that

this was done with reference to a very similar system on the Cantera website derived by Yuanjie Jiang[3]. Their code, found as a Cantera example, was utilized as a template and the derivations of equations in their report were also referenced. Since the system is 1D, the governing equations can be simplified to include only the axial (*z*) direction. The del operators in the governing equations noted above simply reduce to $\frac{d}{dz}$. The conservation of mass reduces to:

$$u\frac{d\rho}{dz} + \rho\frac{du}{dz} = \frac{P'}{A_c}\sum^{K_g}\dot{s}_{k,g}W_{k,g} \qquad \textbf{Eq. 9}$$

The conservation of momentum is reduced to **Eq. 10**. For a 1D reactor with turbulent flow through a circular duct, the viscous drag term can be approximated to be the last of **Eq. 10**.

$$2\rho u\frac{du}{dz} + u^2\frac{d\rho}{dz} = -\frac{dp}{dz} - 0.0791\frac{\rho u^2}{2}Re^{-1/4} \qquad \textbf{Eq. 10}$$

The conservation of energy is simplified to be **Eq. 11** where the overall heat transfer coefficient, $\hat{h}$ is defined as **Eq. 12,** using the Dittus-Boelter equation for the Nusselt relation.

$$\rho u A_c c_p\frac{dT}{dz} = -A_c\sum^{K_g}\dot{\omega}_k W_k h_k - P'\sum^{K_g}h_k\dot{s}_k W_k + P'\hat{h}(T_{wall} - T_{gas}) \qquad \textbf{Eq. 11}$$

$$\hat{h} = \frac{1}{h_{conv}} = (\frac{Nuk_g}{D})^{-1} = (\frac{k_g}{D}0.023Re_D^{0.8}Pr^{0.4})^{-1} \qquad \textbf{Eq. 12}$$

The conservation of species simplifies to:

$$\rho u A_c\frac{dY_k}{dz} = -Y_k p'\sum^{K_g}\dot{s}_{k,g}W_{k,g} + \dot{\omega}_k W_k A_c + P'\dot{s}_{k,g}W_{k,g}, \qquad k = 1...N_g \qquad \textbf{Eq. 13}$$

Finally, the equation of state and the algebraic constraints of surface species production, gas and surface phase species fraction sums are all the same as before (**Eqs. 5-8**).

These governing equations are then inputted into the residual function for the DAE solver. The format to do so is to subtract the LHS of each equation from both sides such that each equation equals 0. Note that since there are $4 + N_g + N_s$ variables (u, $\boldsymbol{\varrho}$, T, p, y_k [k=1...N_g], Z_k [k=1...N_s]), the amount of equations will be the same. The next step is to determine the initial values of each constant, variables, and their derivatives. The constants such as diameter and length are arbitrarily decided and thus can be given any values. For this example, D = 15cm and L = 1.5m. Further, conditions of the reactor are semi-arbitrary. The initial inlet temperature

should be 300K (around room temperature) while the Pt surface temperature was 900K[1]. The initial pressure was set to be 1 atm. The initial velocity was set to 10 m/s, ensuring a turbulent Reynolds number. Finally, the gas composition was set to be the following mole fractions: H_2 = 0.2, O_2 = 0.1, and N_2 = 0.7. The other constants and initial variables were determined through calculations and/or through Cantera's Solution or Interface methods. Once set, it is possible to solve for the derivative values at z = 0 by using Numpy's[N4] linear matrix solver (np.linalg.solve). The matrix containing the coefficients of the derivatives, *a*, is of dimensions (4+N_g) x (4+N_g) while the matrix containing the constants (RHS of equations) is a (4+N_g) x 1. The variable matrix is shown as **Eq. 14**.

$$X = \left(\frac{du}{dz}|_{z=0} \quad \frac{d\rho}{dz}|_{z=0} \quad \frac{dT}{dz}|_{z=0} \quad \frac{dp}{dz}|_{z=0} \quad \frac{dy_1}{dz}|_{z=0} \quad \cdots \quad \frac{dy_{N_{gas}}}{dz}|_{z=0}\right)^T \qquad \textbf{Eq. 14}$$

Displayed in a $aX = b$ form, the following equation must be solved.

$$u_0\frac{d\rho}{dz}|_{z=0} + \rho_0\frac{du}{dz}|_{z=0} = \frac{P'}{A_c}\sum^{K_g}\dot{s}_{k,g}W_{k,g} \qquad \textbf{Eq. 15}$$

$$\rho_0 u_0 A_c\frac{dy_k}{dz}|_{z=0} = -Y_{k,0}P'\sum^{K_g}\dot{s}_{k,g}W_{k,g} + \dot{\omega_k}W_k A_c + \dot{s}_{k,g}W_{k,g}P' \qquad \textbf{Eq. 16}$$

$$2\rho_0 u_0\frac{du}{dz}|_{z=0} + u_0^2\frac{d\rho}{dz}|_{z=0} + \frac{dp}{dz}|_{z=0} = -\frac{32u_0\mu}{D^2} \qquad \textbf{Eq. 17}$$

$$-RT\frac{d\rho}{dz}|_{z=0} + \bar{W_0}\frac{dp}{dz}|_{z=0} - p_0\frac{\sum^{K_g}\frac{dy_k}{dz}|_{z=0}/W_{k,g}}{(\sum^{K_g}Y_k/W_{k,g})^2} = 0 \qquad \textbf{Eq. 18}$$

Once solved for, the DAE solver is run starting at z = 0 to z = L. The mass fractions of the gas species are shown in **Fig. 1** while the surface site fractions are shown in **Fig. 2**. Note that as seen by the production of H_2O and depletion of H_2 and O_2, it is most efficient to end the reactor at just about 0.5m along the reactor. The mixture has completely reacted to form water. As the fluid continues to travel past 0.5m in the reactor, a small amount of water binds to about 1% of the surface site fraction. Other amounts of water bind and break into O(S) or OH(S) which combine to take up about 12.5% of the surface site fraction. No matter how small, this shows that the most efficient length of reactor given the initial speed and diameter of the PFR is 0.5m. Both increasing diameter and increasing speed causes the most effective length to increase.

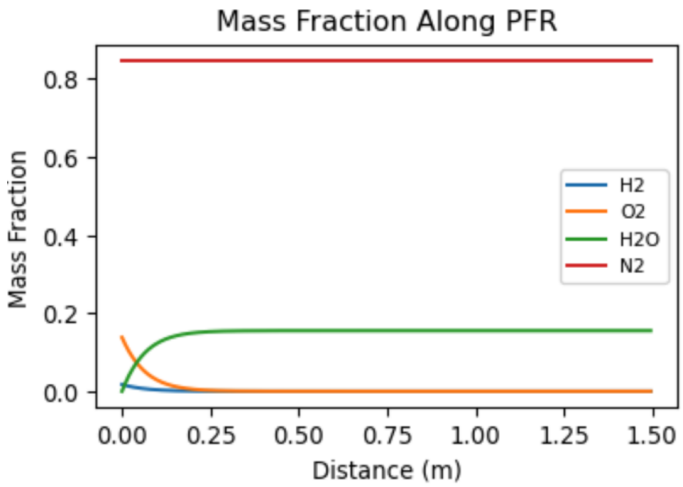


Fig 1: The gas species mass fraction in the fluid along the length of the reactor

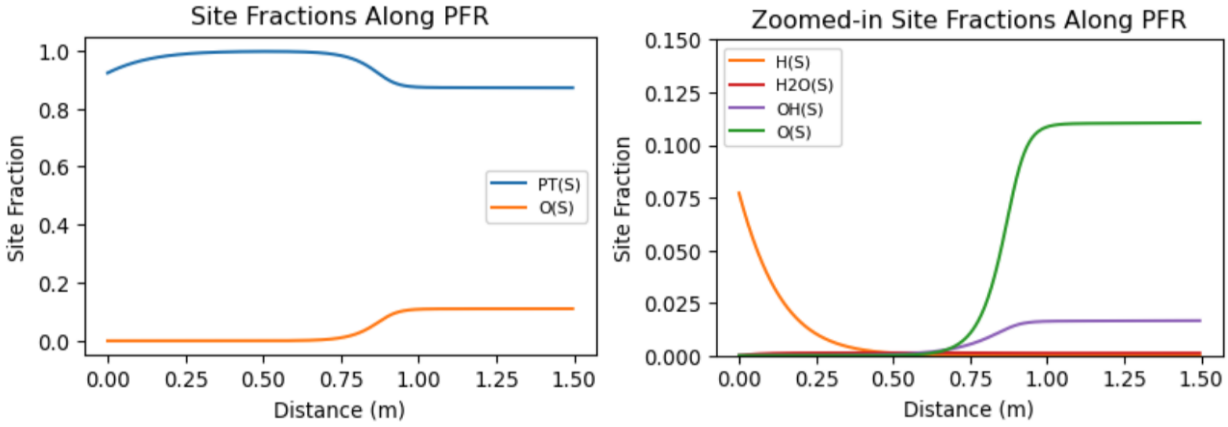


Fig 2: **2a (left)** shows the major surface site fraction species over the length of the reactor. **2b (right)** shows the species that take up <15% of the surface sites along the length of the reactor.

The Damkohler number is a relationship between the rate of reaction and the rate of transport. It is difficult to exactly find the correct relation with the right variables especially with surface chemistry involved. However, it should be known that a Da << 1 assumes that the fluid shoots right through the reactor without much reaction, while a Da >> 1 assumes that the fluid reacts very quickly within the reactor. A large value of Da means that a large amount of the reactor is useless. In the example given above, the Da would be larger than 1 because the reaction occurs early in the reactor. By increasing the rate of transport (increasing velocity) or by decreasing the rate of reaction (increasing diameter), the Da will grow closer to 1.

Some minor observations are that the velocity slightly decreases along the reactor as expected due to the shear stress from the wall (**App. Fig. 1**). The pressure significantly decreases during the combustion reaction, but begins the increase as the fluid continues along the reactor (**App. Fig. 2**). The density slightly increases along the reactor (**App. Fig 3**). Finally, the temperature barely changes, as expected, since the volumetric flow rate is relatively fast and the overall heat transfer coefficient of the fluid is small (**App. Fig 4**).

2D Reactor with Heterogeneous Chemistry

Next, a 2D reactor with heterogeneous chemistry was investigated. This is an increase in dimension from the PFR, meaning that the MoL must be utilized. There are several other differences between the two reactors. The first is that the reactor is no longer a tube, but a 2D rectangle such that the height is shorter than the length. This means that boundary conditions at the floor (Pt surface for catalysis) and the ceiling (non-permeable boundary) must be considered. Second, the flow will be laminar and flowing down the length of the reactor (this time we denote the direction *x*, whereas the direction perpendicular to flow is *z*). Thus, the assumption of perfect mixing in the direction perpendicular to the large convection component is no longer valid. Since the flow is laminar, this allows for small velocities and diffusion can occur in the *z* direction; though dispersion does not occur in the *x* direction because convection is much greater than diffusion. Third, for simplicity, change in temperature will be ignored. Fourth, homogeneous reactions will be neglected because the fluid temperature was too low relative to the temperature needed for combustion without catalysis in the PFR.

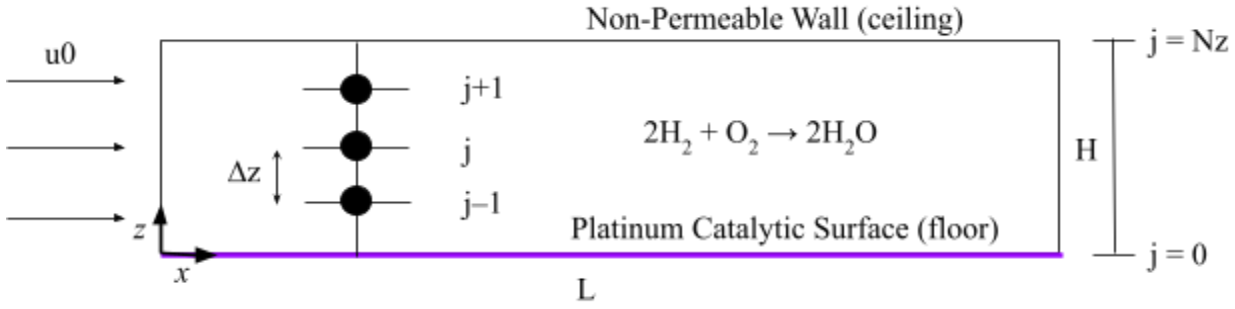


Fig. 3: Diagram of 2D reactor. L and H are the length and heights of the reactor respectively. The flow enters from the left side (x=0) with a uniform velocity u_0. The discretization of the height is indexed by 'j' where j=0 at the catalytic floor (x=0) and j=Nz at the ceiling.

The governing equations must be simplified to be in 2D. Note that the following governing equations will include both transport and surface equations, but different simplifications will need to be made depending on boundary conditions. Since this system must be prepared for MoL, one dimension must be discretized while the other is integrated as usual. Here, the vertical, *z*-direction will be discretized since it has second order derivatives and the fluid flow is dominated by transport in the *x*-direction. The discretization of first and second order derivatives are shown below in **Eq. 19** and **20**. The variable 'f' will act as a placeholder in these equations, j represents the index of the discretized area in the *z*-direction.

$$\frac{\partial f}{\partial z} = \frac{f^{j+1} - f^{j-1}}{2\triangle z} \quad \textbf{Eq. 19}$$

$$\frac{\partial^2 f}{\partial z^2} = \frac{f^{j+1} - 2f^{j} + f^{j-1}}{\triangle z^2} \quad \textbf{Eq. 20}$$

The conservation of mass is derived to be:

$$\rho\frac{du}{dx} + u\frac{d\rho}{dx} = -v(\frac{\rho^{j+1} - f^{\rho-1}}{2\triangle z}) - \rho(\frac{v^{j+1} - v^{j-1}}{2\triangle z}) + \frac{1}{\triangle z}\sum^{N_g} \dot{s_k} W_k \quad \textbf{Eq. 21}$$

The conservation of momentum in the *x* and *z* directions are shown in **Eq. 22** and **Eq. 23** respectively.

$$\rho(u\frac{du}{dx} + v(\frac{u^{j+1} - f^{u-1}}{2\triangle z})) = -\frac{dp}{dx} + \mu(\frac{u^{j+1} - 2u^{j} + u^{j-1}}{\triangle z^2}) \quad \textbf{Eq. 22}$$

$$\rho(u\frac{dv}{dx} + v(\frac{v^{j+1} - v^{j-1}}{2\triangle z})) = -(\frac{p^{j+1} - p^{j-1}}{2\triangle z}) + \mu(\frac{v^{j+1} - 2v^{j} + v^{j-1}}{\triangle z^2}) \quad \textbf{Eq. 23}$$

The conservation of species is shown as:

$$\rho u\frac{dy_k}{dx} = -v(\frac{y_k^{j+1} - y_k^{j-1}}{2\triangle z}) - \rho D_k(\frac{y_k^{j+1} - 2y_k^{j} + y_k^{j-1}}{\triangle z^2}) \quad \textbf{Eq. 24}$$

The conservation of energy is just simplified to be $T = T_0$ (**Eq. 25**) and thus can be excluded from the governing equations for this assumption. Similarly to the PFR, the equation of state and algebraic constraints remain the same as the original governing equations.

These equations represent the most general forms, but they must be solved at every discretized location (j) along the *z* axis. The fluid can be broken into 3 "zones" which have different forms/simplifications of the governing equations due to boundary conditions or

assumptions. At the boundaries, any discretization becomes only dependent on the index j above (for floor) or below (for ceiling) rather than a center discretization.

$$\frac{\partial f}{\partial z}|_{j=floor/ceiling} = \frac{f^{j\pm1} - f^{j}}{\triangle z} \quad \textbf{Eq. 25}$$

$$\frac{\partial^2 f}{\partial z^2}|_{j=floor/ceiling} = \frac{2f^{j\pm1} - 2f^{j}}{\triangle z^2} \quad \textbf{Eq. 26}$$

The first zone is the layer of fluid directly above the catalytic surface (indexed as j=0 in **Fig. 3**). This is the only place where chemistry is taken into account (recall homogeneous reactions are neglected and there is only heterogeneous catalysis). Here the simplifications are that horizontal velocity $u^{j=0} = 0$ due to no-slip boundary conditions. **Eqs. 5-8, 21-23**, and **25** are thus transformed by substituting in **Eq. 25, 26** and setting all $u^{j=0}$ to 0. The convective and diffusive mass fluxes of the fluid in this layer are balanced by the net production rates of the fluid by surface reactions[4]. This relationship is displayed in **Eq. 27**, where **n** is the unit outward-pointing normal vector to the surface (only concerned with the *z*-direction)**.** Here, the mass diffusion velocity from the surface denoted $\mathbf{V_k}$ is defined by **Eq. 28**.

$$\mathbf{n} \cdot \left[\rho Y_k \left(\mathbf{V_k} + \mathbf{u}\right)\right] = \dot{s}_k W_k \qquad (k = 1, \ldots, K_g) \text{ [4]} \quad \textbf{Eq. 27}$$

$$\mathbf{V}_k = -\frac{1}{Y_k} D_{kj} \nabla Y_k \text{ [4]} \quad \textbf{Eq. 28}$$

Usually, **n**·**u** would be defined as the Stefan velocity due to the net mass flux between the surface in the gas. However, since the Stefan velocity is inherently transient and this study is at steady state, that term reduces to zero. The conservation of species equation is thus simplified to:

$$-\rho D_k \left(\frac{y_k^{j+1} - 2y_k^{j}}{\triangle z^2}\right) = \dot{s}_k W_k \quad \textbf{Eq. 29}$$

The next zone is denoted the "middle" and is the area in the reactor that has no surface boundaries (j = 1 to j = N_z – 1). The middle has the same equations as the general ones (**Eq. 21-25** and **5, 7**) but with the deletion of any surface reaction term.

The last zone is the layer of fluid right below the ceiling surface and will be denoted the "ceiling" (indexed j = Nz). Similar to the floor, the no-slip boundary condition applies and $u^{j=Nz}=0$. The two major differences are that 1) there is no reaction at the ceiling so those terms

disappear, and 2) there is no velocity or flux in the z direction ($\frac{\partial v}{\partial z} = 0$, $\frac{\partial^2 v}{\partial z^2} = 0$, $\frac{\partial y_k}{\partial z} = 0$) so those terms also disappear. The largest fundamental change to an equation is for the conservation of species which is simplified to:

$$y_k^{j=Nz} = y_k^{j=Nz-1} \qquad \textbf{Eq. 30}$$

With these equations defined, the residual function for the DAE solver can be made. The input variable and derivative arrays must be a 1D array, but it can best be visualized and unpacked as a 2D one.

```
vec = [[u_0,  v_0,  rho_0,  T_0,  p_0,  y(k=0)_0,  ... y(k=Nk)_0]
       [u_1,  v_1,  rho_1,  T_1,  p_1,  y(k=0)_1,  ... y(k=Nk)_1]
       ...
       [u_Nz, v_Nz, rho_Nz, T_Nz, p_Nz, y(k=0)_Nz, ... y(k=Nk)_Nz]
       [Z_0, . . .  Z_Ns]]

dvec = [[dudx_0,  dvdx_0,  drhodx_0,  dTdx_0,  dpdx_0,  dydx(k=0)_0,  ... dydx(k=Nk)_0]
        ...
        [dudx_Nz, dvdx_Nz, drhodx_Nz, dTdx_Nz, dpdx_Nz, dydx(k=0)_Nz, ... dydx(k=Nk)_Nz]
        [dZdx_0,  . . .    dZdx_Ns]]
```

Fig. 4: Format of reshaped variable array ('vec') and reshaped derivative array ('dvec').

Note that the last row of each array is the surface site fraction for each surface species which is extracted individually to simplify reshaping of the original array, but is present in this way for better visualization. The length of each array is $N_z(4 + N_k) + Ns$ since there are 4 state variables + mass fractions of each gas phase species at every discretized location of z + the surface site fractions of the surface phase species.

The residual function, similar to the 1D PFR simply consists of unpacking the input variable and derivative arrays and looping through all of the indexes of the *z* discretization where conditionals separate the governing equations for the 3 different zones. Each equation is properly indexed into a residual equation array which has the same length as the variable arrays.

The most difficult part, then, becomes determining the initial values for the variables and derivatives since the arrays have gotten much more complex and are all coupled together. Horizontal velocity, u_0 is set to be such that the flow is laminar relative to the height of the reactor. The pressure, gas temperature and catalytic surface temperature are set to the same as the 1D PFR (1 atm, 300K, 900K). The challenge is determining properties at the catalytic boundary which influence vertical velocity v_0 and mass fraction y_k which change in *z* such that they are

consistent with the other properties in the system. The proper way to determine these initial conditions would be to analyze a Von Mises Transformation and solve for them.

It became apparent that it would be best to first solve the system in a simpler form, then increase the complexity later. To do so, the assumption was made that vertical velocity, v, is negligible, thus setting v = 0 for any of the previous equations solved for the 2D reactor case. In this way, mass transfer in the vertical direction only occurs through diffusion of gas phase species. The governing equations are thus significantly reduced in complexity as are the calculations of initial conditions. The code for this case can be found in **App. Code 2**. Results are yet to be found due to a currently unsolved bug in the code which causes the residual function to believe the indexing of an equation does not match the dimension of the residual equation array. The reason for this is unclear because the indexing was analyzed and the allocated amount of dimensions in the array matches the input array dimension, but the error message displays a different value than expected. Thus, this code is still under investigation.

Conclusion and Future Considerations

This project aimed to utilize a DAE solver for Python to solve the governing equations associated with a 1D PFR and a 2D reactor involving fluid flow of a H_2/O_2 mixture and Pt heterogeneous catalytic combustion. The simplification of these equations to lower dimensionality and to obey certain types of situations were discussed. The simulation of the 1D PFR was successful in solving its respective governing equations along the axial direction and outputted expected results. The much more complicated derivation of the 2D reactor governing equations were discussed where the MoL was utilized. Due to the increase in complexity, the resulting code must still be worked on to obtain a successful output. Once the no-vertical-velocity case of this problem is solved the next step would be to reimplement the vertical velocity. By doing so, the Von Mises Transformation would need to be applied such that the initial conditions are self-consistent. Another approach would also be to implement the conservation of energy term such that changes in temperature can be tracked throughout the mixture in the reactor. Finally, reactor effectiveness can be determined with measurement of Da.

References

[1] “catalytic_combustion.py”. Cantera. 2021. [Web April 20221] https://cantera.org/examples/python/surface_chemistry/catalytic_combustion.py.html

[2] Goldsmith, Franklin. “ENGN 2910Q- Lecture Notes”. Brown University. 2021.

[3] Jiang, Yuanjie. “1D Plug Flow Reactor Model with Surface Chemistry”. Cantera. 2018. [Web April 2021] https://cantera.org/examples/jupyter/reactors/1D_pfr_surfchem.ipynb.html.

[4] Kee, Robert J., Coltrin, Michael E, Glaborg, Peter, and Zhu, Huayang. *Chemically Reacting Flow: Theory, Modeling, and Simulation*, 2nd Edition. John Wiley & Sons, Inc. 2018.

[5] Samir Hamdi et al. (2007) Method of lines. Scholarpedia, 2(7):2859. http://www.scholarpedia.org/article/Method_of_lines#Elements_of_the_MOL

Numerical Tools

[N1] Benny, Pavol Kišon, James Tocknell, florian98765, Claas Abert, Marc-Antonio Bisotti, … Martin Robinson. (2020, January 23). bmcage/odes: Release 2.6.1 (Version v2.6.1). Zenodo. http://doi.org/10.5281/zenodo.3625628

[N2] David G. Goodwin, Raymond L. Speth, Harry K. Moffat, and Bryan W. Weber. Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes. https://www.cantera.org, 2021. Version 2.5.1. doi:10.5281/zenodo.4527812

[N3] Malengier et al., (2018). ODES: a high level interface to ODE and DAE solvers. Journal of Open Source Software, 3(22), 165, https://doi.org/10.21105/joss.00165

[N4] Harris, C., Millman, S., Gommers, P., Cournapeau, E., Taylor, J., Berg, N., Kern, R., Picus, S., Kerkwijk, M., Haldane, J., Wiebe, P., Gérard-Marchant, K., Reddy, T., Weckesser, H., & Gohlke, T. (2020). Array programming with NumPy. Nature, 585, 357–362.

[N5] Hindmarsh, A., Brown, P., Grant, K., Lee, S., Serban, R., Shumaker, D., & Woodward, C. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Transactions on Mathematical Software (TOMS), 31(3), 363–396.

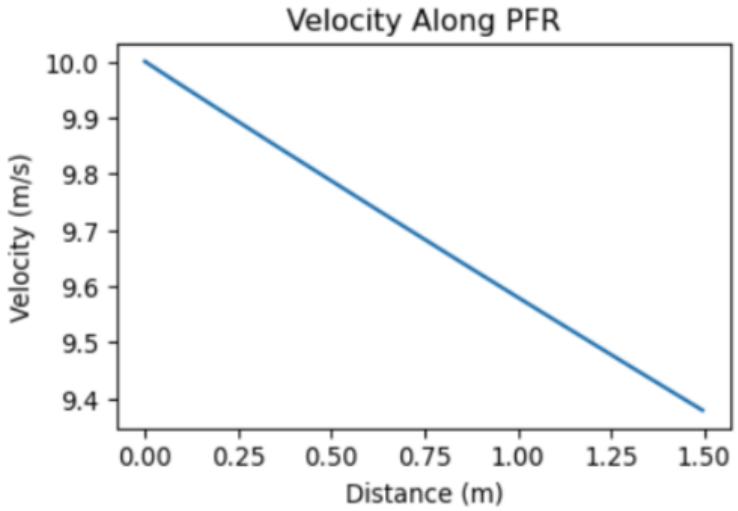
[N6] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55 (publisher link)

Appendix

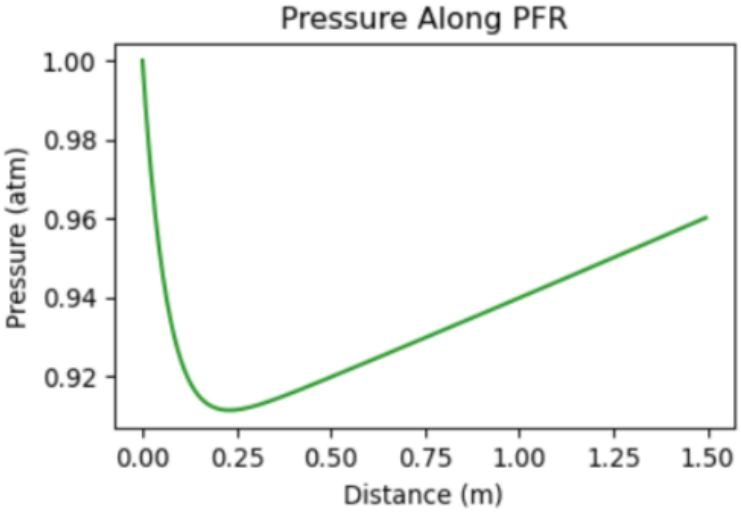
Symbols

ρ	density	λ	conduction HT coeff
$\vec{v}$	velocity vector	h_k	enthalpy of species k
u	velocity (x)	$\vec{j_k}$	diffusion of species k
v	velocity (z)	D_k	diffusion of species k in the total mixture
p	pressure	$c_{p,k}$	heat capacity of species k
T	temperature	W_k	molar weight of species k
$\vec{g}$	gravitational acceleration	$\dot{s_k}$	molar rate of production of species k from surface
μ	viscosity	$\dot{\omega}$	molar rate of production of species k (homog. rxn)
$\bar{c_p}$	average heat capacity	y_k	mass fraction of species k in gas mixture
N_g or N_k	number of gas phase species	Z_k	surface site fraction of species k
N_s	number of surface phase species	$\hat{h}$	overall heat transfer coefficient
D	diameter	h_{conv}	convectional heat transfer coefficient
P’	catalytic perimeter	k_g	gas conduction heat transfer coefficient
A	cross sectional area	Re	Reynolds number
Nu	Nusselt number	Pr	Prandtl number

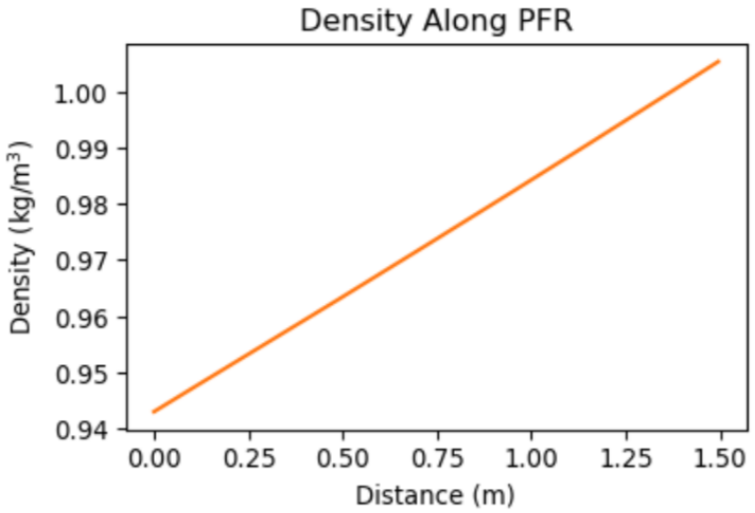
Appendix Figures



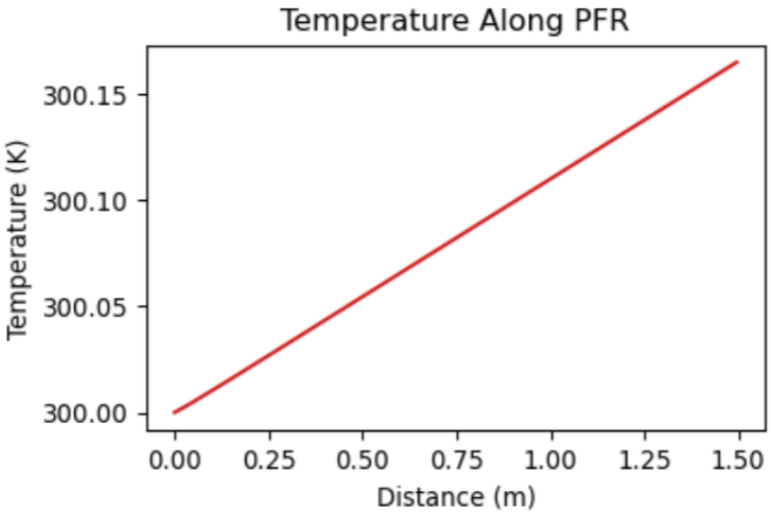
App. Fig. 1: The velocity along the PFR.



App. Fig. 2: The pressure along the PFR



App. Fig. 1: The density along the PFR.



App. Fig. 3: The temperature along the PFR.

Appendix Code

The code for the 1D PFR with Heterogeneous Chemistry and partial code for 2D Reactor with Heterogeneous Chemistry are found below.

1D PFR Simulation of H_2/O_2 System with Pt Catalyzed Combustion

Kento Abeywardane

A reworking of Yuanjie Jiang's work for 1D PFR Model with Surface Chemistry [1][2]

In [1]:
```
import numpy as np
import scipy.linalg
import matplotlib.pyplot as plt
import cantera as ct
from scikits.odes import dae
%matplotlib inline
print("Running Cantera Version: " + str(ct.__version__))
```

```
Running Cantera Version: 2.5.1
```

Reactor Conditions

Set the reactor conditions:

- initial composition
- reactor dimensions
- initial gas temperature and pressure
- surface temperature
- load gas and surface cantera objects

In [2]:
```
#-------------------------------------------------------
# composition of the inlet premixed gas for the hydrogen/air case
comp = 'H2:0.2, O2:0.1, N2:0.7'
Tinlet = 300.0  # inlet temperature, K
Tsurf = 900.0  # surface temperature, K
p = ct.one_atm # pressure, Pa

# pipe parameters
diameter = 0.2 # diameter, m
P = np.pi*diameter # perimeter, m
A = np.pi*diameter**2/4 # area, m
L = 1.5 # m

# load H2-O2 gas
gas = ct.Solution('ptcombust.yaml', 'gas', transport_model='Mix')
gas.TPX = Tinlet, p, comp # set initial gas conditions
# load Pt surf
surf_phase = ct.Interface('ptcombust.yaml', 'Pt_surf', [gas])
surf_phase.TP = Tsurf, p
# get to steady state conditions for surface coverage
surf_phase.advance_coverages(100.0) # advance 100s
```

Governing Equations in Residual Form for DAE Solver

$$R[0] = u\frac{d\rho}{dz} + \rho\frac{du}{dz} - \frac{P'}{A_c}\sum^{K_g}\dot{s}_{k,g}W_{k,g} \tag{1}$$

$$R[1] = Y_{k=0} - 1 \tag{2}$$

$$R[2:(2+K_g)] = \rho u A_c\frac{dY_k}{dz} + Y_k p'\sum^{K_g}\dot{s}_{k,g}W_{k,g} - \dot{\omega}_k W_k A_c - \dot{s}_{k,g}W_{k,g}P' \tag{3}$$

$$R[1+K_g] = \rho u A_c c_p\frac{dT}{dz} + A_c\sum^{K_g}\dot{\omega}_k W_k h_k + P'\sum^{K_g}h_k\dot{s}_k W_k - P'\hat{h}(T_{wall} - T_{gas}) \tag{4}$$

$$R[2+K_g] = 2\rho u\frac{du}{dz} + u^2\frac{d\rho}{dz} + \frac{dp}{dz} + 0.0791\frac{\rho u^2}{2}Re^{-1/4} \tag{5}$$

$$R[3+K_g] = p\bar{W} - \rho RT \tag{6}$$

$$R[(4+K_g):(5+K_g+K_s)] = \dot{s}_{k,s} \tag{7}$$

$$R[(4+K_g+index\ of\ species\ max\ coverage] = \sum^{K_s}Z_{k,s} - 1 \tag{8}$$

In [3]:

```
def residual(z, vec, dvec, result):
    """
    vec = [u, rho, T, p, yk, Zk]
    dvec = [dudz, drhodz, dTdz, dpdz, dykdz, dZkdz]
    """

    ## unpack the values as temporary variables
    u = vec[0]
    rho = vec[1]
    T = vec[2]
    p = vec[3]
    yk = vec[4:4+N]
    Zk = vec[4+N:]

    dudz = dvec[0]
    drhodz = dvec[1]
    dTdz = dvec[2]
    dpdz = dvec[3]
    dykdz = dvec[4:4+N]
    dZkdz = dvec[4+N:]

    # set initial conditions for gas and surface objects
    gas.set_unnormalized_mass_fractions(yk)
    gas.TP = T, p
    surf_phase.set_unnormalized_coverages(Zk)
    surf_phase.TP = Tsurf, p

    #-------------------------------------------------------

    ## temporary variables for each integration
    coverages = surf_phase.coverages
    sdotg = surf_phase.get_net_production_rates(1) # gas phase net production rates from surface, kmol/m^3/s
    sdots = surf_phase.get_net_production_rates(0) # surface phase net production rates, kmol/m^2/s
    wdot = gas.net_production_rates # gas phase net production rates in gas, kmol/m^3/s
    W = gas.molecular_weights # molecular weights array, kg/kmol
    h = gas.enthalpy_mass # enthalpy normalized by mass, J/kg

    ## for Tau wall
    mu = gas.viscosity # viscosity, Pa-s
    Re = rho*u*diameter/mu # Reynolds Number
    tau_wall = rho*u**2/2*0.0791*Re**(-1/4) # shear stress from wall, turbulent
    # tau_wall = rho*u**2./2.*16./Re # shear stress from wall, laminar

    ## for Heat Transfer Coefficient (h_hat)
    k_g = gas.thermal_conductivity # thermal conductivity
    cp = gas.cp # heat capacity
    alpha = k_g/rho/cp # thermal diffusivity
    nu = mu/rho # kinematic viscosity
    Pr = nu/alpha # Prandtl number
    Nu = 0.023*Re**(4/5)*Pr*(0.4) # Nusselt correlation, turbulent circular duct
    #Nu = 3.66 + (0.065*Re*Pr*diameter/L)/(1.+0.04*(Re*Pr*diameter/L)**(2./3.)) # Nusselt correlation, laminar
    h_conv = Nu*k_g/diameter # convectional heat transfer coefficient
    h_hat = 1./h_conv # heat transfer coefficient (h_hat)

    #-------------------------------------------------------

    # Conservation of Mass
    result[0] = u*drhodz + rho*dudz - P/A*np.sum(sdotg*W)

    # Conservation of Species
    for k in range(gas.n_species):
        result[1+k] = (rho*u*A*dykdz[k] + yk[k]*P*np.sum(sdotg*W)
                       - P*sdotg[k]*W[k]
                       - A*wdot[k]*W[k])

    # Conservation of Energy
    result[1+gas.n_species] = rho*u*A*cp*dTdz + P*np.sum(sdotg*W*h) - h_hat*P*(Tsurf - T) - A*np.sum(wdot*W*h)

    # Conservation of Momentum
    result[2+gas.n_species] = 2.*rho*u*dudz + u**2.*drhodz + tau_wall*P/A

    # Equation of State
    result[3+gas.n_species] = rho - gas.density

    # -------------------------------------------------------

    ## Algebraic Constraints
    # replace Nitrogen conservation of species equation with N-1 conservation of species constraint
    index1 = 1+gas.n_species-1
    result[index1] = np.sum(yk) - 1

    # production rates of surface species
```

```
    for k in range(surf_phase.n_species):
        result[4+gas.n_species+k] = sdots[k]

    # replace a constraint with the condition sum(Zk) = 1 for the largest site fraction species
    index2 = np.argmax(coverages)
    result[4+gas.n_species+index2] = np.sum(coverages) - 1

    # ----------------------------------------------------------
```

Initial Values

Solve for Initial Values of Derivatives with np.linalg.solve

$$x = \left(\frac{du}{dz}|_{z=0} \quad \frac{d\rho}{dz}|_{z=0} \quad \frac{dT}{dz}|_{z=0} \quad \frac{dp}{dz}|_{z=0} \quad \frac{dy_1}{dz}|_{z=0} \quad \cdots \quad \frac{dy_{N_{gas}}}{dz}|_{z=0} \right)^T \tag{9}$$

$$\tag{10}$$

$$ax = b \tag{11}$$

$$u_0 \frac{d\rho}{dz}|_{z=0} + \rho_0 \frac{du}{dz}|_{z=0} = \frac{P'}{A_c} \sum^{K_g} \dot{s}_{k,g} W_{k,g} \tag{12}$$

$$\rho_0 u_0 A_c \frac{dy_k}{dz}|_{z=0} = -Y_{k,0} P' \sum^{K_g} \dot{s}_{k,g} W_{k,g} + \dot{\omega}_k W_k A_c + \dot{s}_{k,g} W_{k,g} P' \tag{13}$$

$$2\rho_0 u_0 \frac{du}{dz}|_{z=0} + u_0^2 \frac{d\rho}{dz}|_{z=0} + \frac{dp}{dz}|_{z=0} = -\frac{32 u_0 \mu}{D^2} \tag{14}$$

$$-RT \frac{d\rho}{dz}|_{z=0} + \bar{W}_0 \frac{dp}{dz}|_{z=0} - p_0 \frac{\sum^{K_g} \frac{dy_k}{dz}|_{z=0} / W_{k,g}}{(\sum^{K_g} Y_k / W_{k,g})^2} = 0 \tag{15}$$

In [4]:

```
# calculate initial conditions for dvec with np.linalg.solve
# a = coefficient for [u', rho', T', p', yk', Zk=0]
# b = RHS constant of each conservation eqn
N = gas.n_species # number of gas phase species

# initial values
u0 = 10.0 # velocity, m/s
rho0 = gas.density # density, kg/m^3
sdotg0 = surf_phase.get_net_production_rates(1) # gas phase net production rates from surface, kmol/m^3/s
sdots0 = surf_phase.get_net_production_rates(0) # surface phase net production rates, kmol/m^2/s
wdot0 = gas.net_production_rates # gas phase net production rates in gas, kmol/m^3/s
W0 = gas.molecular_weights # molecular weights array, kg
indexes = [0,3,-1] # indexes of initial composition
W0_avg = np.mean(gas.molecular_weights[indexes]) # average weight
h0 = gas.enthalpy_mass # enthalpy array, J/Kg
Zk0 = surf_phase.coverages # surface coverage fractions
##for Tau wall
mu0 = gas.viscosity # viscosity
Re0 = rho0*u0*diameter/mu0 # Reynolds number
tau_wall0 = rho0*u0**2/2*0.0791*Re0**(-1/4) # shear wallturbulent
#tau_wall0 = rho0*u0**2./2.*16./Re0 # laminar
# for h_hat, Heat Transfer Coefficient
k_g0 = gas.thermal_conductivity # thermal conductivity
cp0 = gas.cp # heat capacity
alpha0 = k_g0/rho0/cp0 # thermal diffusivity
nu0 = mu0/rho0 # kinematic viscosity
Pr0 = nu0/alpha0 # Prandtl number
Nu0 = 0.023*Re0**(4/5)*Pr0*(0.4) # Nusselt correlation, turbulent circular duct
# Nu0 = 3.66 + (0.065*Re0*Pr0*diameter/L)/(1.+0.04*(Re0*Pr0*diameter/L)**(2./3.)) # Nussult correlation, laminar
h_conv0 = Nu0*k_g0/diameter # convectional heat transfer coefficient turbulent
h_hat0 = 1./h_conv0 # heat transfer coefficient (U)
# ---------

# a coefficient matrix
# initialize a
a = np.zeros((4+N, 4+N))
# CoM
a[0] = np.hstack((rho0, u0, np.zeros(2+N)))
# CoE
a[1] = np.hstack((0,0,rho0*u0*A*cp0, np.zeros(1+N)))
# NS
a[2] = np.hstack((2*rho0*u0, u0**2, 0, 1, np.zeros(N)))
# Conservation of Species
for i in range(N):
    a[3+i,4+i] = rho0*u0*A
```

```
coef = np.zeros(N)
# Eq. of state derivative coef
for k in range(N):
    coef[k] = gas.P/W0[k]/np.power(np.sum(gas.Y/W0),2)
a[-1] = np.hstack((0, ct.gas_constant*Tinlet, 0, -W0_avg, coef))

# ---------
# b values
# initialize b matrix
b = np.zeros((4+N))
# CoM
b[0] = P/A*np.sum(sdotg0*W0)
# CoE
b[1] = -P*np.sum(sdotg0*W0*h0) - h_hat0*P*(Tsurf-Tinlet) - A*np.sum(wdot0*W0*h0)
# NS
b[2] = tau_wall0*P/A
# Conservation of Species
for k in range(N):
    b[3+k] = -P*sdotg0[k]*W0[k] - A*wdot0[k]*W0[k] + gas.Y[k]*P*np.sum(sdotg0[k]*W0[k])

# solve the matrix equation
partial_dvec0 = np.linalg.solve(a,b)

# pack initial variable and derivative array
vec0 = np.hstack((u0, rho0, Tinlet, p, gas.Y, Zk0))
dvec0 = np.hstack((partial_dvec0, np.zeros(surf_phase.n_species)))
```

DAE Solver

In [5]:

```
# get the index's of the algebraic constraint equations
algebraic_vars_idx = np.hstack((1, np.arange(4+gas.n_species,4+gas.n_species+surf_phase.n_species,1))).tolist()

# DAE solver settings
solver = dae(
    'ida',
    residual,
    atol=1e-8,  # absolute tolerance for solution
    rtol=1e-8,  # relative tolerance for solution
    algebraic_vars_idx=algebraic_vars_idx,
    max_steps=5000,
    one_step_compute=True,
    old_api=False
)

# solve the DAE
zaxis = []
solution = []
state = solver.init_step(0.0, vec0, dvec0)
while state.values.t < L:
    zaxis.append(state.values.t)
    solution.append(state.values.y)
    state = solver.step(0.01)

# convert list to array
zaxis = np.array(zaxis)
solution = np.array(solution)
```

In [6]:

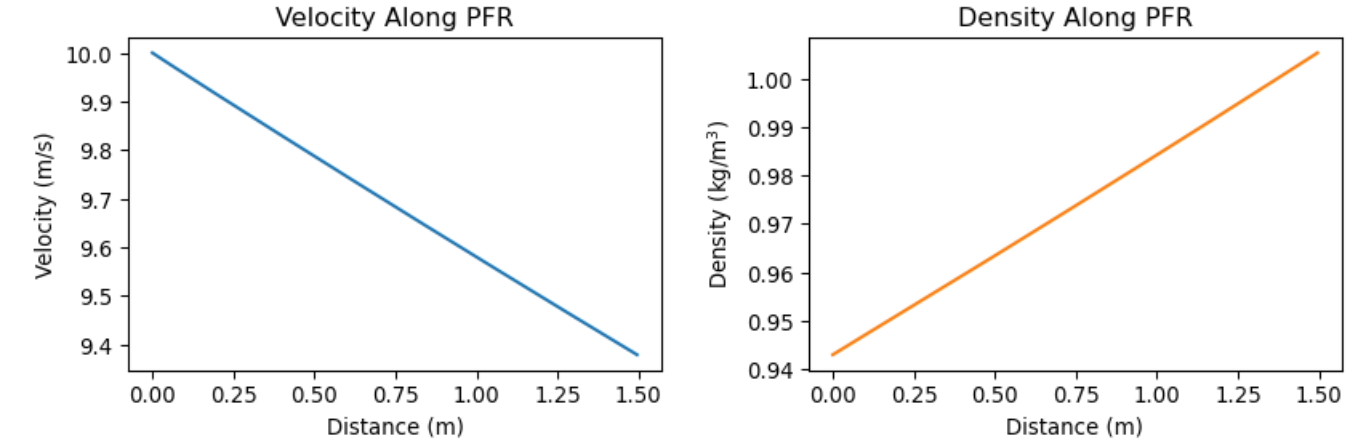
```
# solution arrays
usol = solution[:,0] # velocity (m/s)
rhosol = solution[:,1] # density (kg/m^3)
Tsol = solution[:,2] # Temp (K)
psol = solution[:,3] # pressure (Pa)
yksol = solution[:,4:4+N] # yk
Zksol = solution[:,4+N:] #Zk
```

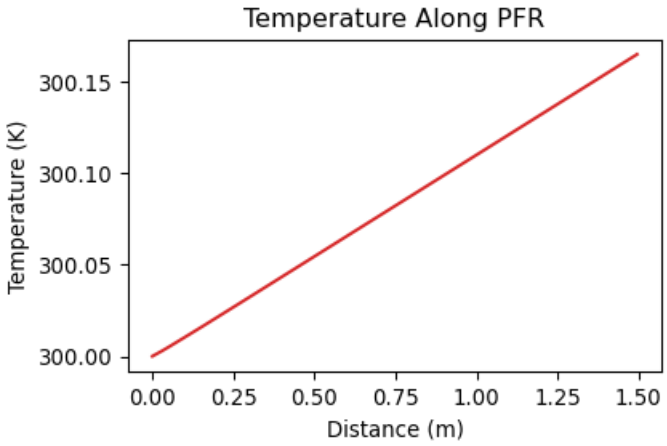
Plot Results

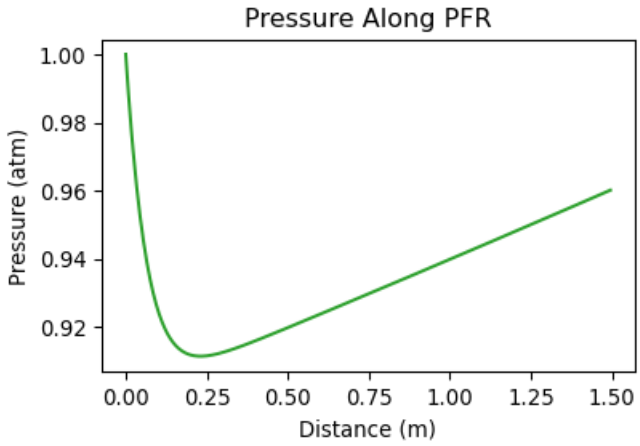
In [7]:

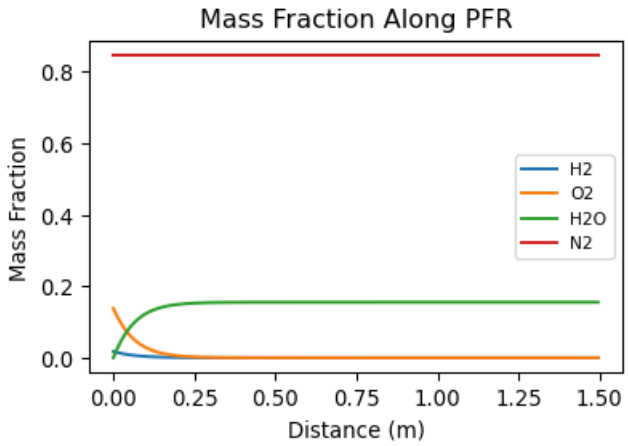
```
## plot
f, ax = plt.subplots(4,2, figsize=(9,12), dpi=96)

# plot gas velocity along the flow direction
ax[0,0].plot(zaxis, usol, color='C0')
ax[0,0].set_xlabel('Distance (m)')
ax[0,0].set_ylabel('Velocity (m/s)')
ax[0,0].set_title('Velocity Along PFR')
```

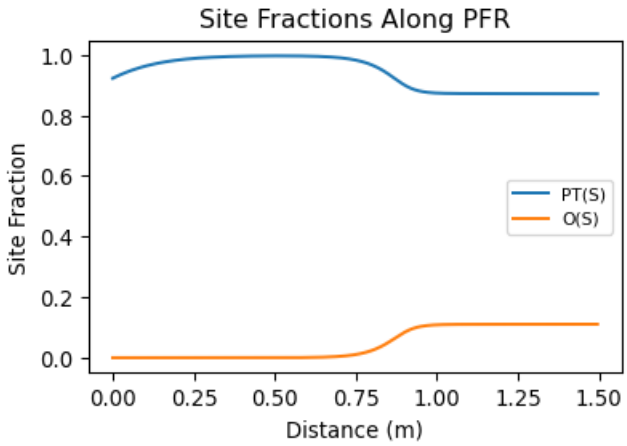
```
# plot gas density along the flow direction
ax[0,1].plot(zaxis, rhosol, color='C1')
ax[0,1].set_xlabel('Distance (m)')
ax[0,1].set_ylabel('Density ($\mathregular{kg/m^3}$)')
ax[0,1].ticklabel_format(axis='y', style='sci', scilimits=(-2,2))  # scientific notation
ax[0,1].set_title('Density Along PFR')

# plot the temperature profile along the flow direction
ax[1,0].plot(zaxis, Tsol, color='C3')
ax[1,0].set_xlabel('Distance (m)')
ax[1,0].set_ylabel('Temperature (K)')
ax[1,0].set_title('Temperature Along PFR')

# plot the pressure of the gas along the flow direction
ax[1,1].plot(zaxis, psol/ct.one_atm, color='C2')
ax[1,1].set_xlabel('Distance (m)')
ax[1,1].set_ylabel('Pressure (atm)')
ax[1,1].set_title('Pressure Along PFR')

# plot major and minor gas species separately
minor_idx = []
major_idx = []
for k, name in enumerate(gas.species_names):
    mean = np.mean(yksol[:,k])
    if mean >= 0.001:
        major_idx.append(k)

# plot major gas species along the flow direction
for j in major_idx:
    ax[2,0].plot(zaxis, yksol[:,j], label=gas.species_names[j])
ax[2,0].legend(fontsize=8, loc='best')
ax[2,0].set_xlabel('Distance (m)')
ax[2,0].set_ylabel('Mass Fraction')
ax[2,0].set_title('Mass Fraction Along PFR')

# plot the site fraction of the surface species along the flow direction
for i,name in enumerate(surf_phase.species_names):
    if 'C' not in name and np.mean(Zksol[:,i])>0.01:
        ax[2,1].plot(zaxis, Zksol[:,i], label=name)
    if 'C' not in name and np.mean(Zksol[:,i])>0.001 and np.mean(Zksol[:,i])<0.5:
        if name == 'H(S)':
            ax[3,1].plot(zaxis, Zksol[:,i], label=name, color='#ff7f0e')
        elif name == 'O(S)':
            ax[3,1].plot(zaxis, Zksol[:,i], label=name, color='#2ca02c')
        elif name == 'OH(S)':
            ax[3,1].plot(zaxis, Zksol[:,i], label=name, color='#9467bd')
        elif name == 'H2O(S)':
            ax[3,1].plot(zaxis, Zksol[:,i], label=name, color='#d62728')
ax[2,1].legend(fontsize=8)
ax[2,1].set_xlabel('Distance (m)')
ax[2,1].set_ylabel('Site Fraction')
ax[2,1].set_title('Site Fractions Along PFR')

ax[3,1].set_ylim([0,0.15])
ax[3,1].legend(fontsize=8, loc='best')
ax[3,1].set_xlabel('Distance (m)')
ax[3,1].set_ylabel('Site Fraction')
ax[3,1].set_title('Zoomed-in Site Fractions Along PFR')

f.tight_layout()
plt.show()
```

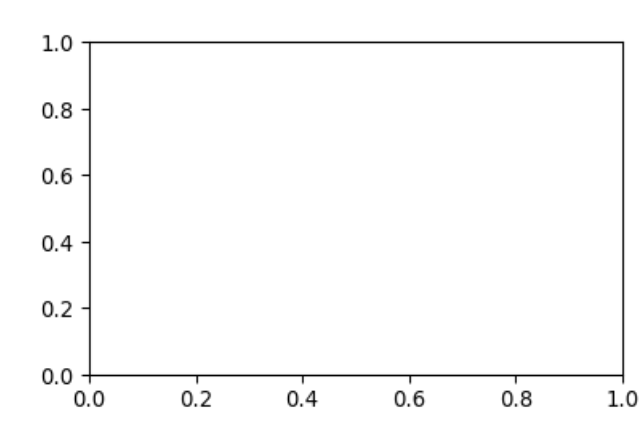


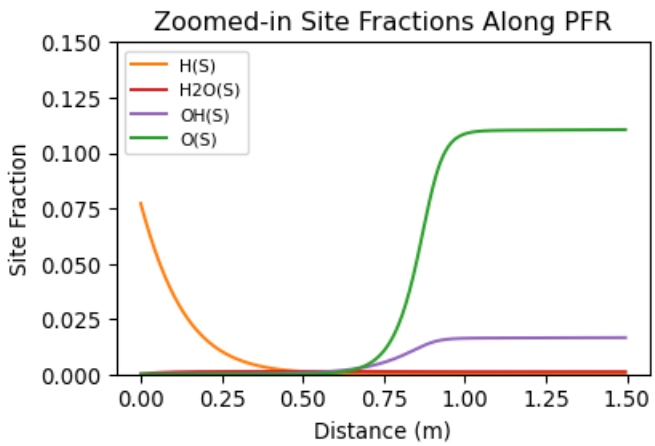












Citations:

Yuanjie Jiang, July 2018

[1] https://cantera.org/examples/jupyter/reactors/1D_pfr_surfchem.ipynb.html \ [2] https://github.com/yuj056/yuj056.github.io/blob/master/Week1/yuj056_github_io.pdf

[3] https://cantera.org/examples/python/surface_chemistry/catalytic_combustion.py.html

2D Reactor with Heterogenous Chemistry

No Vertical Velocity

Do not run without stopping quickly or it will take up too much memory

In [ ]:
```
import numpy as np
import scipy.linalg
import matplotlib.pyplot as plt
import pylab
import matplotlib.gridspec as gridspec
import cantera as ct
from scikits.odes import dae
%matplotlib inline
print("Running Cantera Version: " + str(ct.__version__))
```

Reactor Conditions

In [ ]:
```
#-----------------------------------------------------------
# composition of the inlet premixed gas for the hydrogen/air case
comp = 'H2:0.6, O2:0.8, N2:0.2'
Tinlet = 300.0  # inlet temperature, K
Tsurf = 900.0  # surface temperature, K
p0 = ct.one_atm # pressure, Pa
# pipe parameters
L = 1. # m
H = 0.1 # m
# load H2-O2 gas
gas = ct.Solution('ptcombust.yaml', 'gas', transport_model='Mix')
gas.TPX = Tinlet, p0, comp # set initial gas conditions
# load Pt surf
surf = ct.Interface('ptcombust.yaml', 'Pt_surf', [gas])
surf.TP = Tsurf, p0
# get to steady state conditions
surf.advance_coverages(100.0)

Nk = gas.n_species
Ns = surf.n_species
Nz = 51
delz = H/(Nz-1)
W = gas.molecular_weights
```

Residual DAE solver function

In [ ]:
```
def residual(x, nvec, ndvec, result):
    vec = nvec[Ns:].reshape([Nz,4+Nk]) # from 1D array to 2D array for easier indexing
    dvec = ndvec[Ns:].reshape([Nz,4+Nk]) # from 1D array to 2D array for easier indexing
    """
    vec = [[u_0, rho_0, T_0, p_0, yk_0]
           [u_1, rho_1, T_1, p_1, yk_1]
           ...
           [u_Nz, rho_Nz, T_Nz, p_Nz, yk_Nz]
           [Z_0...Z_Ns]]

    dvec = [[dudx_0, drhodx_0, dTdx_0, dpdx_0, dykdx_0]
            ...
            [dudx_Nz, drhodx_Nz, dTdx_Nz, dpdx_Nz, dykdx_Nz]
            [dZdx_0...dZdx_Ns]]
    """

    #-----------------------------------------------------------
    # loop through values in z direction
    for j in range(0,Nz):
        # define variables
        u = vec[j,0]
        udown = vec[j-1,0]
        rho = vec[j,1]
        rhodown = vec[j-1,1]
        T = vec[j,2]
        Tdown = vec[j-1,2]
        p = vec[j,3]
        pdown = vec[j-1,3]
```

```
yk = vec[j,4:4+Nk] # yk[j,k] where j is index of Z and k is species
ykdown = vec[j-1,4:4+Nk]
Zk = nvec[:Ns]

if j != Nz-1:
    uup = vec[j+1,0]
    rhoup = vec[j+1,1]
    Tup = vec[j+1,2]
    pup = vec[j+1,3]
    ykup = vec[j+1,4:4+Nk]

# define x derivatives
dudx = dvec[j,0]
drhodx = dvec[j,1]
dTdx = dvec[j,2]
dpdx = dvec[j,3]
dykdx = dvec[j,4:4+Nk]
dZkdx = ndvec[:Ns]
gas.set_unnormalized_mass_fractions(yk)
gas.TP = Tinlet, p

# gas
W = gas.molecular_weights # kg/kmol

wdot = gas.net_production_rates
h = gas.enthalpy_mass
mu = gas.viscosity
Dk = gas.mix_diff_coeffs_mass
Wmean = gas.mean_molecular_weight


# floor layer
if j == 0:
    ## temporary variables for each integration, surface
    # surface
    surf.set_unnormalized_coverages(Zk)
    surf.TP = Tsurf, p
    coverages = surf.coverages
    sdotg = surf.get_net_production_rates(1) # kmol/m^3/s
    sdots = surf.get_net_production_rates(0) # kmol/m^2/s


    # conservation of mass
    result[0] = np.sum(sdotg*W)
    # conservation of momentum x
    result[1] = (dpdx - mu*(2.*uup)/delz**2.)
    # conservation of energy
    result[2] = T - Tinlet
    # equation of state
    result[3] = rho - gas.density
    # conservation of species N-1
    result[4:4+Nk] = -rho*Dk*(ykup - yk)/delz - sdotg*W
    ####
    # Algebraic Constraints
    # replace nitrogen with sum of gas species mass fractions = 1
    result[4+Nk-1] = np.sum(yk) - 1.
    # species production
    result[4+Nk:4+Nk+Ns] = sdots
    # surface site fraction sum = 1
    index = np.argmax(coverages)
    result[4+Nk+index] = np.sum(coverages) - 1.

# middle
elif j > 0 and j < Nz-1:
    middlei = 4+Nk+Ns
    # conservation of mass
    result[middlei+(j-1)*middlei] = u*drhodx + rho*dudx
    # conservation of momentum x
    result[middlei+(j-1)*middlei+1] = (rho*u*dudx
                                       + dpdx
                                       - mu*((uup-2.*u+udown)/delz**2.))
    # conservation of energy
    result[middlei+(j-1)*middlei+2] = T - Tinlet
    # eqn of state
    result[middlei+(j-1)*middlei+3] = rho - gas.density
    # conservation of gas species N-1
    result[middlei+(j-1)*middlei+4:middlei+(j-1)*middlei+4+Nk] = (- rho*u*dykdx
                                                                  - rho*Dk*(ykup-2.*yk+ykdown)/delz**2.)
    # conservation of gas species fractions
    index1 = middlei+(j-1)*middlei+4+Nk-1
    result[index1] = np.sum(yk) - 1.
```

```
        # ceiling layer (implicit boundary conditions)
        elif j == Nz-1:
            # new starting index for ceiling
            ceili = middlei+(Nz-2-1)*middlei+4+Nk
            # conservation of mass (no slip and dvdz = 0)
            result[ceili] = u
            # conservation of momentum, x
            result[ceili+1] = (dpdx - mu*(2.*udown/delz**2.))
            # conservation of energy
            result[ceili+2] = T - Tinlet
            # equation of State
            result[ceili+3] = rho - gas.density
            # conservation of species N-1
            result[ceili+4:ceili+4+Nk] = yk - ykdown
            ## algebraic
            # conservation of gas species fractions
            result[ceili+4+Nk-1] = np.sum(yk) - 1.
```

Determine Initial Conditions

In [ ]:

```
# initial conditions
u0 = 0.5 #m/s
rho0 = gas.density
indexes = [0,3,-1] # indexes of initial composition
W0_avg = np.mean(gas.molecular_weights[indexes])
mu0 = gas.viscosity

# middle
vecmiddle0 = np.hstack((u0, rho0, Tinlet, p0, gas.Y))
vecmiddle_total0 = np.hstack((u0, rho0, Tinlet, p0, gas.Y))
for j in range(2, Nz-1):
    vecmiddle_total0 = np.vstack((vecmiddle_total0, vecmiddle0))
dvecmiddle_total0 = np.zeros((Nz-2,4+Nk))

# ceiling
pprime = -mu0*u0/delz**2.
vecceil0 = np.hstack((0, rho0, Tinlet, p0, gas.Y))
dvecceil0 = np.hstack((0, 0, 0, pprime, np.zeros(Nk)))

# surf
vecsurf0 = np.hstack((0, rho0, Tinlet, p0, gas.Y))
dvecurf0 = np.hstack((0, 0, 0, pprime, np.zeros(Nk)))

# Z
Zk0 = surf.coverages
vec0_part = np.vstack((vecsurf0, vecmiddle_total0, vecceil0)).reshape((1,Nz*(4+Nk)))[0]
vec0 = np.hstack((Zk0, vec0_part))

dvec0_part = np.vstack((dvecurf0, dvecmiddle_total0, dvecceil0)).reshape((1,Nz*(4+Nk)))[0]
dvec0 = np.hstack((np.zeros(Ns), dvec0_part))
```

Call DAE Solver

In [ ]:

```
# get the index's of the algebraic constraint equations
algebraic_vars_idx = np.hstack((np.arange(5+Nk-1,5+Nk+Ns)))
for j in range(Nz):
    if j > 0 and j < Nz-1:
        middlei = 5+Nk+Ns
        algebraic_vars_idx = np.hstack((algebraic_vars_idx, np.arange(middlei+(j-1)*middlei+5+Nk-1,middlei+(j-1)*mid
ceili = middlei+(Nz-2-1)*middlei+5+Nk+Ns
algebraic_vars_idx = np.hstack((algebraic_vars_idx, np.arange(ceili+5+Nk-1,ceili+5+Nk+Ns)))
algebraic_vars_idx = algebraic_vars_idx.tolist()

# solve DAE
solver = dae(
    'ida',
    residual,
    atol=1e-8,  # absolute tolerance for solution
    rtol=1e-8,  # relative tolerance for solution
    algebraic_vars_idx=algebraic_vars_idx,
    max_steps=5000,
    one_step_compute=True,
    old_api=False
)
```

```
# xaxis = []
# solution = []
# state = solver.init_step(0.0, vec0, dvec0)
# while state.values.t < 0.1:
#     print(state.values.t)
#     xaxis.append(state.values.t)
#     solution.append(state.values.y)
#     state = solver.step(0.01)

# xaxis = np.array(xaxis)
# solution = np.array(solution)

times = np.linspace(0,0.1,10)
solution = solver.solve(times, vec0, dvec0)
```

In [ ]:

```
solution.message
```