University of Massachusetts, Lowell
Department of Mechanical Engineering

Finite Volume Simulation of Acetone Cracking in a Non-Ideal Plug Flow Reactor

Patrick Heng

MECH 5200

Prof. D.J. Willis

04/30/25

Abstract

Reactive transport phenomena is a fundamental framework for the modeling of chemical reactors. While these models have a strong theoretical basis, the resulting partial differential equations that describe the phenomena are highly non-linear and have significant coupling between each other. This leads to 'stiff' differential equations that are expensive or unstable to integrate, even for relatively simple reactions. In the following, we explore a finite volume based solver for 1D problems and develop an algorithm to iteratively solve the Concentration, U momentum, and Temperature (CUT) equations. As a case study, we further develop a model for a plug flow reactor with axial dispersion and test the solver with conditions simulating the thermal cracking of acetone. This reaction has a highly temperature dependent rate constant and thus serves as a reasonable benchmark to test the performance of the CUT algorithm in a moderately non-linear setting.

Academic Integrity

I hereby affirm that the following work submitted is my own and that I have not received help from others outside of this class. Furthermore, I shall reference any resources used outside of class lectures and notes.

– P.V. Heng

Contents

Table 1: Nomenclature

Symbol	Denotes	Units (SI)
c_i	Concentration of species i	$\mathrm{mol \cdot m^{-3}}$
M_i	Molar mass of species i	$\mathrm{mol \cdot kg^{-1}}$
ν_i	Stoichiometric coefficient of species i	1
$C_{p,i} = a_0 + a_1 T + a_2 T^2$	Molar heat capacity of species i	$\mathrm{J \cdot mol^{-1} \cdot K^{-1}}$
r_i	Total reaction rate of species i	$\mathrm{mol \cdot m^{-3} \cdot s^{-1}}$
u	Axial flow velocity	$\mathrm{m \cdot s^{-1}}$
T	Absolute temperature	K
p	Absolute pressure	$\mathrm{J \cdot m^{-3}}$
$\hat{\rho} = \sum_i c_i M_i$	Mixture density	$\mathrm{kg \cdot m^{-3}}$
$\hat{H} = \sum_i T c_i C_{p,i}(T)$	Mixture enthalpy	$\mathrm{J \cdot mol^{-1} \cdot K^{-1}}$
ΔH°_{rxn}	Molar reaction enthalpy at reference temperature	$\mathrm{J \cdot mol^{-1}}$
$K = c_{A,0} \sum_i \Theta_i C_{p,i}(T_0)$	Initial volumetric heat capacity	$\mathrm{J \cdot m^{-3}}$
D	Axial dispersion coefficient	$\mathrm{m^2 \cdot s^{-1}}$
μ	Average mixture viscosity	$\mathrm{kg \cdot m^{-1} \cdot s^{-1}}$
κ	Average mixture thermal conductivity	$\mathrm{J \cdot m^{-1} \cdot s \cdot K^{-1}}$
T_a	Heat exchanger temperature	K
T_{ref}	Thermodynamic reference temperature	K
E_a	Reaction activation energy	$\mathrm{J \cdot mol^{-1}}$
k_0	Arrhenius frequency factor	$\mathrm{s^{-1}}$
R	Universal gas constant	$\mathrm{J \cdot mol^{-1} \cdot K^{-1}}$
$\hat{q}_{rxn}$	Heat of reaction	$\mathrm{J \cdot m^{-3} \cdot s^{-1}}$
$\hat{q}_{ext}$	Heat exchanger heating rate	$\mathrm{J \cdot m^{-3} \cdot s^{-1}}$
h	Heat transfer coefficient	$\mathrm{J \cdot m^{-2} \cdot s^{-1}}$
A	Inner reactor surface area	$\mathrm{m^2}$
V	Inner reactor volume	$\mathrm{m^3}$
x	Distance from reactor entrance	m
L	Total reactor length	m

Table 2: Dimensionless Groups

Symbol	Expression	Denotes
C_A	$\dfrac{c_A}{c_{A,0}}$	Normalized concentration of A
U	$\dfrac{u}{U_0}$	Normalized velocity
θ	$\dfrac{T}{T_0}$	Normalized temperature
H	$\dfrac{\hat{H}}{K}$	Normalized enthalpy
P	$\dfrac{p}{p_0}$	Normalized pressure
ρ	$\dfrac{\hat{\rho}}{\rho_0}$	Normalized density
X	$1 - C_A U$	Conversion of A
β	$\dfrac{-E_a}{RT_0}$	Normalized activation energy
Θ_i	$\dfrac{c_{i,0}}{c_{A,0}}$	Normalized initial concentration of species i
Pe_M	$\dfrac{U_0 L}{D}$	Mass Pèlcet number
Pe_T	$\dfrac{U_0 L K}{\kappa}$	Thermal Pèlcet number
Re	$\dfrac{\rho_0 U_0 L}{\mu}$	Reynolds number
Eu	$\dfrac{p_0}{\rho_0 U_0^2}$	Euler number
Da	$\dfrac{k_A L}{U_0}$	Damköhler Number
Γ	$\dfrac{c_{A,0}}{KT_0}\left(\Delta H^{\circ}_{rxn} - \sum_i \nu_i \int_0^{T_{ref}} C_{p,i}\, dT\right)$	Normalized reference enthalpy
q_{rxn}	$-Da\, e^{\beta/\theta} C_A \left(\Gamma + \dfrac{c_{A,0}}{KT_0} \sum_i \nu_i \int_0^{T} C_{p,i}\, dT\right)$	Normalized heat of reaction
q_{ext}	$\dfrac{hAL}{VKU_0}\left(\dfrac{T_a}{T_0} - \theta\right)$	Normalized heat exchange rate
z	$\dfrac{x}{L}$	Normalized distance

* Subscript 0's denote the variable evaluated at the inlet conditions

PFR Model & Governing Equations

Consider a decomposition reaction of A, which forms products, B and C,

$$aA \rightarrow bB + cC, \tag{1}$$

where a, b, and c are the stoichiometric coefficients of the reaction. This reaction can be carried out in a 'plug flow reactor' (PFR) where the fluid reactant is passed through a tube with elevated pressure and temperature to drive the reaction [1].

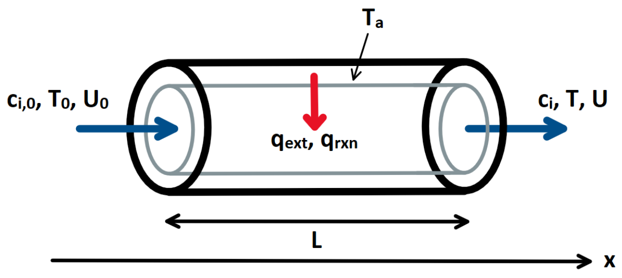


Figure 1: Diagram of plug flow reactor with a shell and tube heat exchanger operating with an outside fluid temperature of T_a. At the inlet, c_i, T, and U are known and specified as Dirichlet boundary conditions. At the outlet, the c_i, T, and U profiles remain undisturbed by the exit, so a no-flux (Neumann) condition is imposed.

A specific example that will be considered to validate the developed numerical method is the thermal cracking of acetone (CH_3COCH_3, A) which is inspired by the problem given in [2]. This reaction produces ketene (CH_2CO, B) which is a precursor to acetic anhydride, an important reactant in organic synthesis. In absence of a catalyst, this is a homogeneous gas phase reaction,

$$\mathrm{CH_3COCH_3} \rightarrow \mathrm{CH_2CO} + \mathrm{CH_4}, \tag{2}$$

which follows first order reaction kinetics. To simplify the analysis of the reactor, we will assume no radial or angular variation of all variables and only consider variation along the x axis. By conservation of mass, momentum, and energy, we have the equations [1] [3],

$$\frac{\partial c_A}{\partial t} + \frac{\partial (u c_A)}{\partial x} = D\frac{\partial^2 c_A}{\partial x^2} + r_A, \tag{3}$$

$$\frac{\partial (\hat{\rho} u)}{\partial t} + \frac{\partial (\hat{\rho} u u)}{\partial x} = \mu\frac{\partial^2 u}{\partial x^2} - \frac{\partial p}{\partial x}, \tag{4}$$

$$\frac{\partial \hat{H}}{\partial t} + \frac{\partial (U\hat{H})}{\partial x} = \kappa\frac{\partial^2 T}{\partial x^2} + \hat{q}_{rxn} + \hat{q}_{ext} + \frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x}. \tag{5}$$

where the main solution variables are the concentration of A, c_A, the fluid velocity, u, and the fluid temperature, T. We have assumed that the axial dispersion coefficient, D, viscosity, μ, and thermal conductivity, κ, are constant and the same for all chemical species. r_A is a reaction term of the form,

$$-\frac{r_A}{a} = k_0 \exp\left(\frac{-E_a}{RT}\right) c_A^{\alpha}, \tag{6}$$

where $\alpha = 1$ for our specific reaction. The volumetric heat released by the reaction, $\hat{q}_{rxn}$, may be modeled by the reaction rate times the reaction enthalpy,

$$\hat{q}_{rxn} = -\frac{r_A}{a}\Delta H_{rxn}(T) = k_0 \exp\left(\frac{-E_a}{RT}\right) c_A^{\alpha}\left(\Delta H^{o}_{rxn} - \sum_i \nu_i \int_0^{T_{ref}} C_{p,i}\, dT + \sum_i \nu_i \int_0^{T} C_{p,i} dT\right), \tag{7}$$

[1]Physically, the diffusion term in each of the equations is small and could be neglected, but they are maintained for numerical stability. Diffusion makes the resulting matrices more diagonally dominant and more suitable to perform iterative methods.

where the heat capacities, $C_{p,i}$, are further given by empirical polynomials [5],

$$C_{p,i}(T) = \left(a_0 + a_1 T + a_2 T^2\right)_i . \tag{8}$$

Finally, the external heating rate, $\hat{q}_{ext}$, may be modeled with a constant ambient temperature heat exchanger,

$$\hat{q}_{ext} = \frac{hA}{V}\left(T_a - T\right), \tag{9}$$

where h is the heat transfer coefficient, A is the heat transfer area, V is the reactor volume, and T_a is the heat exchanger temperature. For computational purposes, it is convenient to non-dimensionalize the governing equations,

$$\frac{\partial C_A}{\partial t} + \frac{\partial\left(UC_A\right)}{\partial z} = \frac{1}{Pe_M}\frac{\partial^2 C_A}{\partial z^2} - Da\, e^{\beta/\theta} C_A, \tag{10}$$

$$\frac{\partial\left(\rho U\right)}{\partial t} + \frac{\partial\left(\rho U U\right)}{\partial z} = \frac{1}{Re}\frac{\partial^2 U}{\partial z^2} - Eu\frac{\partial P}{\partial z}, \tag{11}$$

$$\frac{\partial H}{\partial t} + \frac{\partial\left(UH\right)}{\partial z} = \frac{1}{Pe_T}\frac{\partial^2 \theta}{\partial z^2} + q_{rxn} + q_{ext} + \frac{p_0}{KT_0}\left(\frac{\partial P}{\partial t} + U\frac{\partial P}{\partial z}\right), \tag{12}$$

where the dimensionless variables are defined in the 'Dimensionless Groups' table. While we now have the governing PDE's for our problem, we still must supply an equation of state to solve for the pressure in the reactor. An ideal gas assumption is *not* valid at the operating conditions, however, the dimensionless *ratio* between the inlet and outlet states is approximately ideal for small variations in p and T [1]. With this assumption, we find,

$$P = \frac{\left(1 + \varepsilon X\right)\theta}{U}, \tag{13}$$

$$\rho = \frac{c_{A,0}}{\rho_0}\frac{\sum_i\left(\Theta_i M_i + (\nu_i/a) X M_i\right)}{1 + \varepsilon X}\left(\frac{P}{\theta}\right), \tag{14}$$

$$H = \frac{c_{A,0}}{K}\sum_i\left[\left(\Theta_i - \frac{\nu_i}{a}X\right)C_{p,i}\right]\left(\frac{\theta}{U}\right), \tag{15}$$

where,

$$X = 1 - UC_A, \tag{16}$$

is the conversion of A, and,

$$\varepsilon = -\frac{1}{a}\frac{\sum_i \nu_i}{\sum_i \Theta_i}, \tag{17}$$

is related to the change in moles due to the reaction.

Finite Volume Discretization

Being a system of conservation laws, the finite volume method is a natural approach to solve this problem as it ensures that conservation is satisfied discretely. For the sake of brevity, we will only derive the discretized form of the concentration equation; for a full implementation of each discrete equation, refer to the code in Appendix A. The finite volume formulation is given by integrating the conservation laws over each computational cell, V_i, and dividing by the cell's volume. For the concentration equation,

$$\frac{d}{dt}\left[\frac{1}{V_i}\int_{V_i} C_A\, dz\right] + \frac{1}{V_i}\int_{V_i}\frac{\partial}{\partial z}\left(UC_A\right)dz = \frac{1}{Pe_M}\frac{1}{V_i}\int_{V_i}\frac{\partial}{\partial z}\left(\frac{\partial C_A}{\partial z}\right)dz - \frac{Da}{V_i}\int_{V_i} e^{\beta/\theta} C_A\, dz. \tag{18}$$

Recall that the average of any quantity, ϕ, over a volume, V_i, is defined by,

$$\bar{\phi}_i \coloneqq \frac{1}{V_i}\int_{V_i}\phi_i\, dV, \tag{19}$$

where the overbar denotes the cell averaged quantity. We may also apply the fundamental theorem of calculus to the diffusive and convective flux terms to integrate out the z derivatives. If we assume a unit cross sectional area and uniform grid spacing of Δz, then, $V_i = \Delta z$. Combining these facts yields,

$$\frac{d\bar{C}_{A,i}}{dt} + \frac{(UC_A)_{i+1/2} - (UC_A)_{i-1/2}}{\Delta z} = \frac{1}{Pe_M}\frac{1}{\Delta z}\left(\frac{\partial C_A}{\partial z}\bigg|_{i+1/2} - \frac{\partial C_A}{\partial z}\bigg|_{i-1/2}\right) - Da\left(\overline{e^{\beta/\theta}C_A}\right)\bigg|_i, \tag{20}$$

where the half index notation denotes the evaluation at the cell faces.

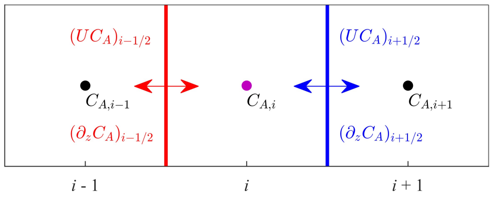


Figure 2: Placement of convective and diffusive fluxes relative to cell i

Observe that all of the manipulations we have performed thus far have been exact. However, to actually solve these equations, we must apply reasonable approximations to the flux and source terms and write them in terms of the cell averages, $\bar{C}_{A,i}$, $\bar{U}_i$, and $\bar{\theta}_i$. The source term allows a natural approximation,

$$\left(\overline{e^{\beta/\theta}C_A}\right)\bigg|_i = e^{\beta/\bar{\theta}_i}\bar{C}_{A,i}. \tag{21}$$

The diffusive fluxes also follow a natural approximation by using finite differences for the derivatives,

$$\frac{\partial C_A}{\partial z}\bigg|_{i+1/2} = \frac{\bar{C}_{A,i+1} - \bar{C}_{A,i}}{\Delta z}, \quad \frac{\partial C_A}{\partial z}\bigg|_{i-1/2} = \frac{\bar{C}_{A,i} - \bar{C}_{A,i-1}}{\Delta z}. \tag{22}$$

The most troublesome term is the convective flux which can lead to instability if discretized incorrectly. Since we will eventually only consider the steady state problem, a linear interpolation between the cell centers will be used. However, this scheme would *not* be stable for a transient problem and upwinding or flux limiting must be considered [4]. Applying the interpolation yields,

$$(UC_A)_{i+1/2} = \frac{\bar{U}_{i+1}\bar{C}_{A,i+1} + \bar{U}_i\bar{C}_{A,i}}{2}, \quad (UC_A)_{i-1/2} = \frac{\bar{U}_i\bar{C}_{A,i} + \bar{U}_{i-1}\bar{C}_{A,i-1}}{2}. \tag{23}$$

Further application of all the discretization schemes then gives,

$$\frac{d\bar{C}_{A,i}}{dt} + \frac{\bar{U}_{i+1}\bar{C}_{A,i+1} - \bar{U}_{i-1}\bar{C}_{A,i-1}}{2\Delta z} = \frac{1}{Pe_M}\frac{\bar{C}_{A,i+1} - 2\bar{C}_{A,i} + \bar{C}_{A,i-1}}{\Delta z^2} - Da\, e^{\beta/\bar{\theta}_i}\bar{C}_{A,i}. \tag{24}$$

Herein, we will only consider the steady state problem where the time derivative vanishes. We also group like-indices, giving,

$$\underbrace{\left[-\bar{U}_{i-1} - \frac{1}{Pe_M\Delta z^2}\right]}_{=W_i}\bar{C}_{A,i-1} + \underbrace{\left[Da\, e^{\beta/\bar{\theta}_i} + \frac{2}{Pe_M\Delta z^2}\right]}_{=PP_i}\bar{C}_{A,i} + \underbrace{\left[\bar{U}_{i+1} - \frac{1}{Pe_M\Delta z^2}\right]}_{=E_i}\bar{C}_{A,i+1} = 0. \tag{25}$$

Similar to finite difference methods, we place the $[W_i, PP_i, E_i]$ vectors on the $[-1, 0, 1]$ diagonals of a sparse $\mathbf{A}$ matrix. This gives us a 'linear-like' form,

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{26}$$

where $\mathbf{x}$ is the vector of $C_{A,i}$ values and $\mathbf{A}$ and $\mathbf{b}$ are *not* necessarily constant and may depend on $C_{A,i}$, U_i, or θ_i [2]. That is, $\mathbf{A}^k = \mathbf{A}(\mathbf{y}^k)$ and $\mathbf{b}^k = \mathbf{b}(\mathbf{y}^k)$, where $\mathbf{y}$ is the vector of *all* unknowns and k is an

[2]The overbar notation is hereby dropped with the understanding that the variables stored are the cell averages.

iteration counter. A common method in CFD codes is 'Picard' type iteration where we guess a value of $\mathbf{y}^k$ and evaluate $\mathbf{A}$ and $\mathbf{b}$ at $\mathbf{y}^k$ to solve for $\mathbf{x}^{k+1}$ [4,6]. Generally, for each transport equation, to update $\mathbf{x}^k$ (which may now represent the vector of $C_{A,i}$, U_i, or θ_i values), we form $\mathbf{A}^k$ and $\mathbf{b}^k$ for that transport equation. From this, we then solve the linear system,

$$\mathbf{A}^k\mathbf{x}^* = \mathbf{b}^k, \tag{27}$$

and update $\mathbf{x}^k$ by,

$$\mathbf{x}^{k+1} = (1-\omega)\mathbf{x}^k + \omega\mathbf{x}^*. \tag{28}$$

ω is a 'relaxation factor' that is often set to a value between 0 and 1 to help establish convergence [4,6]. This is repeated until some converge criterion is met, say,

$$R_x = \frac{||\mathbf{x}^{k+1} - \mathbf{x}^k||}{||\mathbf{x}^{k+1}||} < \delta, \tag{29}$$

where R_x is the relative residual and δ is a small number. The convergence of Picard iterations is usually slow and the error [3] often decreases linearly [6]. An alternative method is to use 'Newton' iteration which uses a linear approximation of $\mathbf{Ax} - \mathbf{b}$ to iteratively solve for the unknowns [4]. With Newton iteration, the error decreases superlinearly and can decrease quadratically at best [6]. However, Newton solvers require a good initial $\mathbf{x}$ to have stable convergence, thus, they are not ideal if the user provides a poor initial guess. Therefore, a reasonable solution algorithm is to use Picard iteration to establish convergence at the first few iterations, then switch to a Newton solver to quickly finish off the error. This is the motivation for the CUT algorithm developed in the following section.

The only discretization left is the boundary conditions, for which we let $\phi = C_A, U,$ or θ, then for each transport equation,

$$\text{at } z = 0: \ \phi = 1, \quad \text{at } z = 1: \ \frac{\partial \phi}{\partial z} = 0. \tag{30}$$

By applying linear extrapolation from the internal cells to each of the boundaries, we may find equivalent discrete conditions on the fluxes. For example, for the concentration equation,

$$\left[U_1 - \frac{2}{Pe_M \Delta z}\right] C_{A,1} = 1 - \frac{2}{Pe_M \Delta z}, \quad \left[\frac{1}{\Delta z}\right] C_{A,NN} - \left[\frac{1}{\Delta z}\right] C_{A,NN-1} = 0, \tag{31}$$

where 1 is the first cell index and NN is the last cell index. Essentially the same conditions may be derived for the momentum and energy equations.

CUT Solution Algorithm

The CUT algorithm stands for 'Concentration, U momentum, Temperature', indicating the order of transport equations to solve during the Picard iterations. The Picard loop begins with the concentration equation since it undergoes the greatest relative change, varying from 0 to 1, and the other equations have large coupling with it. The momentum equation is solved next as a 'corrector' for the P and X fields that are used in the temperature equation. The temperature equation contains the most coupling which is why it is solved last, using all of the most recent updates. At the end of each Picard iteration, a residual is calculated for each field, if the residuals are 'small enough', the loop breaks and the solution is done. Otherwise, the algorithm then checks if the residuals are small enough to use a Newton solver, if they are still too large, the loop continues with another Picard iteration. A flowchart of the algorithm is presented in the following, for a more detailed implementation of the algorithm, refer to the script in Appendix B.

[3]The error is *not* the same as the residual, but the residual is often a good measure of the error.

[4]The details of Newton solvers will not be discussed here, for more information, see MATLAB's documentation for 'fsolve'. Also of interest is 'Jacobian Free Newton Krylov' (JFNK) methods which are commonly used algorithms for large multiphysics problems [7].

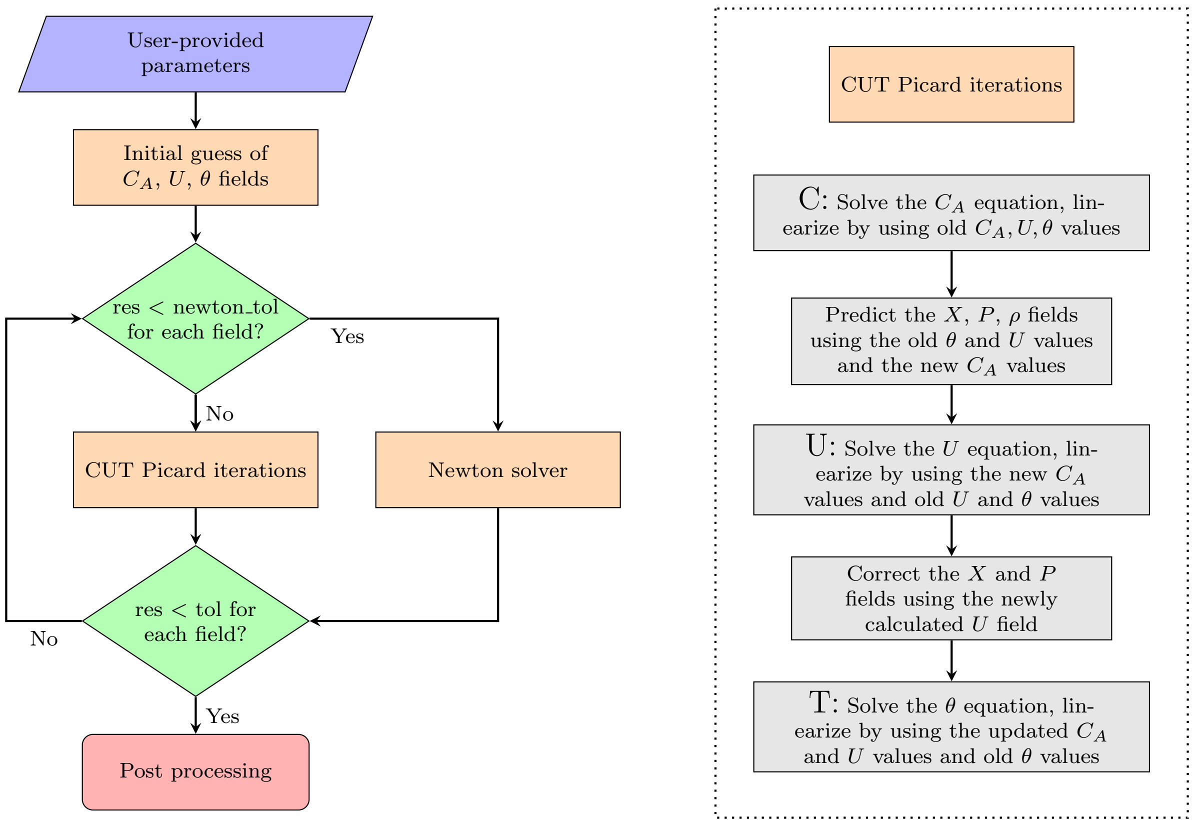

User-provided parameters
Initial guess of C_A, U, θ fields
res < newton_tol for each field?
Yes
No
CUT Picard iterations
Newton solver
res < tol for each field?
No
Yes
Post processing
CUT Picard iterations
C: Solve the C_A equation, linearize by using old C_A, U, θ values
Predict the X, P, ρ fields using the old θ and U values and the new C_A values
U: Solve the U equation, linearize by using the new C_A values and old U and θ values
Correct the X and P fields using the newly calculated U field
T: Solve the θ equation, linearize by using the updated C_A and U values and old θ values

Figure 3: Flowchart of the CUT algorithm for the reaction-convection-diffusion equations

Validation

Let ϕ_i^* be the numerical solution of any of the transport fields on a highly refined mesh. If we perform the same simulation on less refined meshes, we expect that the error of coarser meshes to converge to the refined mesh on the order of $\mathcal{O}(\Delta z^p)$, where p is the order of the convergence. To this end, we define the approximate error, e_i, as,

$$e_i = \phi_i^* - \phi_i.$$

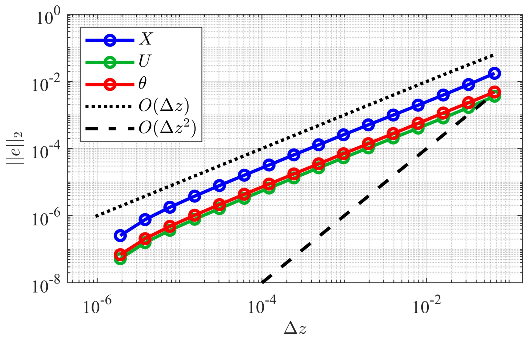
The L_2 norm error may then be approximated as,

$$||e||_2 \approx \left(\sum_{i=1}^{NN} |e_i|^2 \Delta z \right)^{1/2},$$

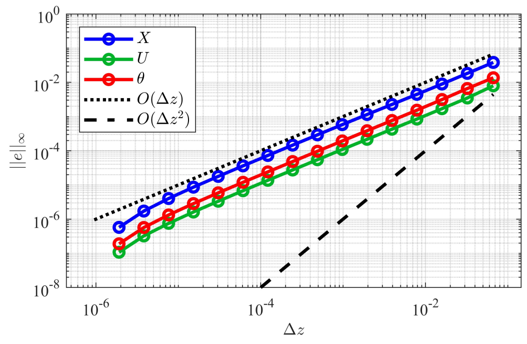
where ϕ_i^* is evaluated at every k steps to make it the same size as ϕ_i. As an absolute error bound, the L_∞ norm error may be calculated by,

$$||e||_\infty = \max(|e_i|).$$

The finest mesh tested was with $NN = 2^{20}$ and $\Delta z = 9.5\text{e-}7$. Successive meshes had NN halved each time until the coarsest mesh with $NN = 2^4$ was tested. These errors may be plotted on a log-log plot and compared to $\mathcal{O}(\Delta z)$ and $\mathcal{O}(\Delta z^2)$ which appear as straight lines.



(a) L_2 error

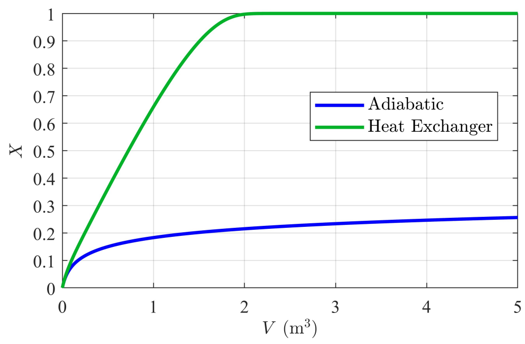


(b) L_∞ error

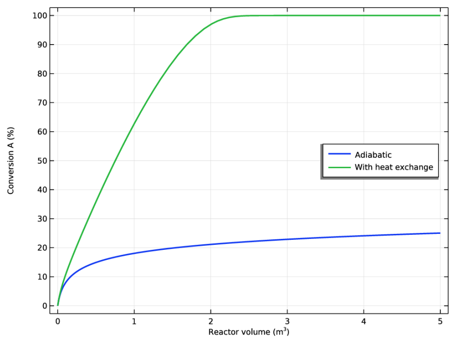
Figure 4: Mesh convergence tests in the L_2 and L_∞ norms

From this, we may conclude that the discretization error is on the order of Δz, meaning that if Δz is halved, then the solution error approximately halves. This error is likely due to the weakly enforced (extrapolated) Dirichlet condition at the left boundary which contaminates the error in the rest of the domain.

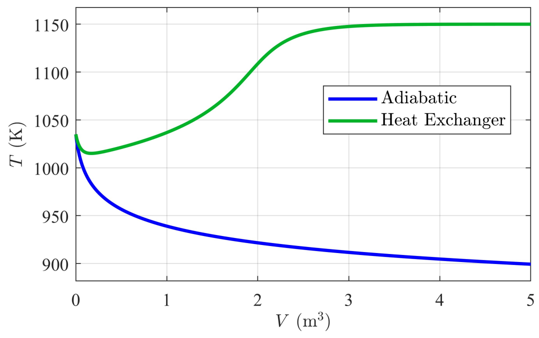
To further validate the results of the solver, the solution was compared to a similar problem presented in [2], which used COMSOL to simulate the PFR. The results are slightly different since the models are not the same. Specifically, the model in [2] does not account for axial dispersion, meaning there is no diffusion, and pressure effects are excluded from the energy equation. However, these terms are small and thus, we should still expect the solutions to be close and have the same general characteristics. The following figures present the conversion, temperature, and reaction rate fields for the CUT scheme and are compared to the plots taken directly from [2]. For information on the specific parameters used, refer to Appendix B.



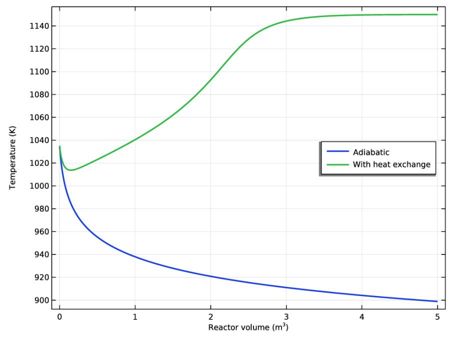
(a) CUT-FV X field



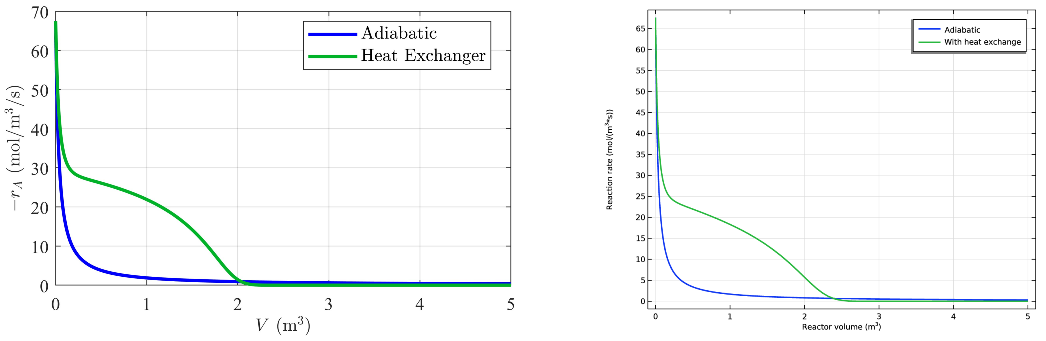
(b) COMSOL X field



(c) CUT-FV T field



(d) COMSOL T field



(a) CUT-FV r_A field **(b)** COMSOL r_A field

Figure 6: Validation of CUT finite volume scheme against COMSOL solution

From this, we verify that the solutions are reasonably close despite the model differences. Notably, we may notice that the inclusion of axial dispersion causes the reaction to progress faster. This can be most clearly seen in (a) and (b) where the heat exchanger curve obtains $X = 1$ slightly before $V = 2$ m^3 for the CUT solution while $X = 1$ slightly after $V = 2$ m^3 for the COMSOL solution. This is physically reasonable since higher dispersion implies more mixing and causes an increase in the reaction rate. Therefore, we may conclude that the diffusion term gives physically reasonable results and is implemented correctly.

Numerical Study of Optimal Operating Conditions

As a brief case study, we may wish to estimate the optimal conditions to operate the reactor under. While there are many parameters in the model, the two with the largest impact are the inlet concentration of acetone, $c_{A,0}$, and the heat exchanger temperature, T_a. We wish to maximize the molar flow rate of ketene, F_B, while minimizing the power consumption of the reactor, $\dot{W}$. As $c_{A,0}$ is increased, more power is required to run the compressor flowing the gas. Suppose we are compressing the gas from $P_i = 1$ atm to the operating conditions, then under the assumption of ideal isothermal compression, the power, $\dot{W}_c$, is estimated by,

$$\dot{W}_c \sim AU_0 c_{A,0} R T_0 \ln\left(\frac{c_{A,0} R T_0}{P_i}\right). \tag{32}$$

The heat exchanger fluid may be assumed to be steam which requires some heating power, $\dot{W}_H$, to get from the boiling point to the operating temperature, T_a. This may be estimated by,

$$\dot{W}_H \sim \dot{m} \int_{373\ \text{K}}^{T_a} C_p \, dT, \tag{33}$$

where C_p is the heat capacity of steam and $\dot{m}$ is the mass flow of steam. $\dot{m}$ is estimated by an overall energy balance between the heat exchanger fluid and the heat released by the reaction,

$$\dot{m} C_p(T_a) \left[T_a - T_{avg}\right] \sim AU_0 c_{A,0} \Delta H_{rxn}(T_0), \tag{34}$$

where $T_{avg} = (T_a + T_0)/2$. Combining these power estimates yields the final cost function,

$$\dot{W} = \dot{W}_c + \dot{W}_H = AU_0 c_{A,0} R T_0 \ln\left(\frac{c_{A,0} R T_0}{P_i}\right) + \frac{2AU_0 c_{A,0} \Delta H_{rxn}(T_0)}{C_p(T_a)\left[T_a - T_0\right]} \int_{T_{ref}}^{T_a} C_p \, dT. \tag{35}$$

By testing different values of $c_{A,0}$ and T_a and calculating the resulting $\dot{W}$ and F_B, we may estimate the behavior of the given reactor. Most of the parameters remain unchanged from the simulation in the validation section, but for a more detailed implementation of the parameters used for the power calculations, refer to Appendix C.

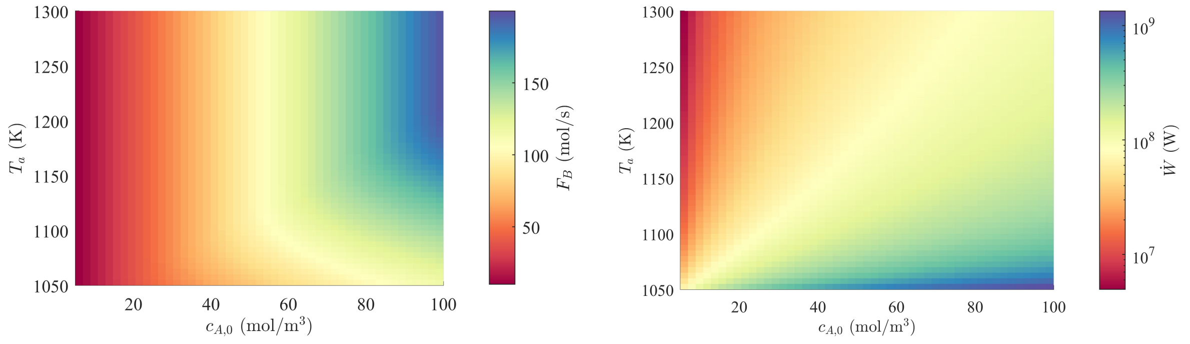


Figure 7: Variation of ketene production, F_B, and PFR power consumption, $\dot{W}$, as a function of inlet acetone concentration, $c_{A,0}$, and heat exchanger temperature, T_a

We observe that low T_a values are unfavorable due to the high steam flow rate required, which is less efficient than just using a low steam flow rate, but heated to a higher temperature. Additionally, the effect of increasing $c_{A,0}$ on $\dot{W}$ is approximately linear, i.e. a 10 times increase in $c_{A,0}$ will approximately result a 10 times increase in $\dot{W}$. From these observations, it is ideal to operate the reactor at the highest temperature possible and at moderate concentrations of acetone. While these are ideal conditions, one must also consider possible safety concerns and side reactions at the more extreme operating conditions. Ketene is extremely reactive and may react with excess acetone at high temperatures. Furthermore, the reaction also produces methane, which means the pressure will increase at the outlet. Therefore, operating at a high temperature and acetone concentration requires a high capital cost to build piping that can withstand these conditions. A more detailed analysis should perform an optimization of the operating and capital costs of the reactor. However, from this brief thermodynamic analysis, we can conclude how we should ideally design and run the reactor.

Conclusion

The cracking of acetone in a plug flow reactor offers a simple case study for the coupling of convection, diffusion, and reaction in a single PDE system. We have developed and validated the CUT algorithm around this model to test its performance on a moderately non-linear problem to ensure its robustness. A natural extension of the algorithm would be to include the transient startup of the reactor, which would not be significantly difficult to implement due to the already implicit nature of the non-linear solver. The main concern with the transient problem would be the convection term which would likely have to include a flux limiter to ensure reasonable solutions. Further investigations may also explore catalytic reactions in packed bed reactors where the rate law is often much more complex and the pressure dependence of the solution becomes more pronounced.

References

[1] Fogler, H.S., *Essentials of Chemical Reaction Engineering.* 2nd Ed., Pearson Education Inc., 2018.

[2] *Non-Isothermal Plug Flow Reactor.* COMSOL Multiphysics.
https://www.comsol.com/model/nonisothermal-plug-flow-reactor-1404

[3] Heng, P.V., *Notes on Transport Phenomena.*
https://github.com/pheng044/Transport_Phenomena

[4] Greenshields, C.J., Weller, H.G., *Notes on Computational Fluid Dynamics: General Principles.* CFD Direct Limited, 2022.

[5] Mcbride, B., Zehe, M., Gordon, S., *NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species.* 2002.

[6] Langtangen, H.P., *Solving Non-Linear ODE and PDE Problems.* Department of Informatics, University of Oslo, 2016.

[7] https://mooseframework.inl.gov/source/systems/NonlinearSystem.html#JFNK

[8] https://www.mathworks.com/help/matlab/ref/spdiags.html

[9] https://www.mathworks.com/help/optim/ug/fsolve.html

Appendix A - Functions

```
% ------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% 1D stencil coefficients for non-dimensionalized PFR problem
%  -> concentration equation
% ------------------------------------------------------------------

function [A,f] = stencil_coefficients_CA(CA,U,theta,params)

    % Load parameters needed to create coefficients
    Pe_M = params.Pe_M;
    Da = params.Da;
    beta = params.beta;
    dt = params.dt;
    dz = params.dz;
    nodes = params.nodes;

    % Compute stencil coefficients
    W = - U/(2*dz) - (1/Pe_M)/dz^2;
    C =  1/dt + (1/Pe_M)*2/dz^2 + Da*exp(beta./theta);
    E = U/(2*dz) - (1/Pe_M)/dz^2;

    % Return coefficients in convenient form for using spdiags
    diag = [W,C,E];

    % Generate A at internal nodes
    A = spdiags(diag,[-1,0,1],nodes,nodes+1);
    A = A(:,1:nodes);

    % Clear boundary rows
    A(1,:) = 0;
    A(nodes,:) = 0;

    % Left flux boundary
    A(1,1) = U(1) - 2/(Pe_M*dz);
    % Right flux boundary
    A(nodes,nodes) = 1/dz;
    A(nodes,nodes-1) = -1/dz;

    % Forcing function with appropriate boundary conditions
    f = CA/dt;                    % Internal
    f(1) = 1 - 2/(Pe_M*dz);       % Right
    f(nodes) = 0;                 % Left

end

% ------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% 1D stencil coefficients for non-dimensionalized PFR problem
%  -> momentum equation
% ------------------------------------------------------------------
```

```
function [A,f] = stencil_coefficients_U(U,rho,P,params)

    % Load parameters needed to create coefficients
    Re = params.Re;
    Eu = params.Eu;
    dt = params.dt;
    dz = params.dz;
    nodes = params.nodes;

    % Compute stencil coefficients
    W = -rho.*U/(2*dz) - (1/Re)/dz^2;
    C = rho/dt + (1/Re)*2/dz^2;
    E = rho.*U/(2*dz) - (1/Re)/dz^2;

    % Return coefficients in convenient form for using spdiags
    diag = [W,C,E];

    % Generate A at internal nodes
    A = spdiags(diag,[-1,0,1],nodes,nodes+1);
    A = A(:,1:nodes);

    % Clear boundary rows
    A(1,:) = 0; A(nodes,:) = 0;

    % Left flux boundary
    A(1,1) = rho(1)*U(1) - 2/(Re*dz);
    % Right flux boundary
    A(nodes,nodes) = 1/dz;
    A(nodes,nodes-1) = -1/dz;

    % Forcing function with appropriate boundary conditions
    f = U/dt + Eu*([P(2:nodes);0]-[0;P(1:nodes-1)])/(2*dz); %Internal
    f(1) = 1 - 2/(Re*dz);           % Left
    f(nodes) = 0;                   % Right

end

% -------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% 1D stencil coefficients for non-dimensionalized PFR problem
%  -> theta equation
% -------------------------------------------------------------------

function [A,f] = stencil_coefficients_theta(X,U,P,theta,params)

    % Load parameters needed to create coefficients
    cA0 = params.cA0;
    stoich = params.stoich;
    THETA = params.THETA;
    p0 = params.p0;
    Pe_T = params.Pe_T;
    Da = params.Da;
    T0 = params.T0;
    Ta = params.Ta;
    K = params.K;
    beta = params.beta;
    Gamma = params.Gamma;
    N = params.N;
    a1 = params.a1;
    a2 = params.a2;
    a3 = params.a3;
    dt = params.dt;
    dz = params.dz;
    nodes = params.nodes;

    % Calculate enthalpy coefficient such that H*theta = total enthalpy
    AA = THETA'*a1 + THETA'*a2*T0*theta + THETA'*a3*T0^2*theta.^2  ...
            - (stoich'*a1+(stoich'*a2*T0)*theta+(stoich'*a3*T0^2) ...
                *theta.^2).*(X/stoich(1));
    H = (cA0/K)*AA./U;

    % Compute stencil coefficients
    W = -U.*H/(2*dz) - (1/Pe_T)*(1/dz^2);
    C = H/dt + (1/Pe_T)*(2/dz^2) + Da*exp(beta./theta).*(1-X) ...
        .*(cA0*(stoich'*a1+(stoich'*a2*T0/2)...
        *theta+((1/3)*stoich'*a3*T0^2)*theta.^2)/K)./U + N;
    E = U.*H/(2*dz) - (1/Pe_T)*(1/dz^2);

    % Return coefficients in convenient form for using spdiags
    diag = [W,C,E];

    % Generate A at internal nodes
    A = spdiags(diag,[-1,0,1],nodes,nodes+1);
    A = A(:,1:nodes);

    % Clear boundary rows
    A(1,:) = 0; A(nodes,:) = 0;

    % Left flux boundary
    A(1,1) = U(1)*H(1) - 2/(Pe_T*dz);
    % Right flux boundary
    A(nodes,nodes) = 1/dz;
    A(nodes,nodes-1) = -1/dz;

    % Forcing function with appropriate boundary conditions

    % Internal nodes
    f = H.*theta/dt - Da*exp(beta./theta).*(1-X).*Gamma./U + N*Ta/T0 ...
            + (p0/K/T0)*U.*([P(2:nodes);0]-[0;P(1:nodes-1)])/(2*dz);
    f(1) = 1 - 2/(Pe_T*dz);      % Left
    f(nodes) = 0;                % Right

end
```

```
% ---------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% Calculate density field for non-dimensionalized PFR equation
% ---------------------------------------------------------------------

function rho = density(X,theta,P,params)

    % Load parameters needed to calculate density
    cA0 = params.cA0;
    stoich = params.stoich;
    THETA = params.THETA;
    epsilon = params.epsilon;
    MW = params.MW;
    rho_0 = params.rho_0;


    % Calculate, return
    rho = (cA0./(1+epsilon*X)).*(sum(THETA.*MW) ...
            - sum(stoich.*MW)*X/stoich(1)).*P./theta/rho_0;

end

% ---------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% Calculate pressure field for non-dimensionalized PFR equation
% ---------------------------------------------------------------------

function P = pressure(X,U,theta,params)

    epsilon = params.epsilon;
    P = (1 + epsilon*X).*theta./U;

end
```

Appendix B - Test Scripts for Validation

```
% -----------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 3 Mar 2025
% Script to solve the PFR boundary value problem using a non-linear
% finite volume scheme.
% Uses the CUT loop to solve the coupled transport problems:
%   C - Concentration equation
%   U - U momentum equation
%   T - Temperature equation
% which is iteratively solved until convergence
% -----------------------------------------------------------------

close all; clear all; clc;

% ----- INPUTS -----

% Initial concentration of A, cA0 (mol/m^3)
cA0 = 18.8;

% Initial concentrations of other species (mol/m^3)
% i = 1        2       3        4
%     Acetone  Ketene  Methane  Nitrogen (inert)
c0 = [cA0;0;0;0];

% Initial temperature, T0 (K)
T0 = 1035;

% Heat exchanger temperature, Ta (K)
Ta = 1150;

% Reactor length, L (m)
L = 5;

% PFR velocity, U (m/s)
U0 = 2;

% Stoichiometric coefficients [a,b,c,d]
stoich = [-1;1;1;0];

% Molecular weights (kg/mol),
% components arranged in same order as stoich vector
MW = [58.08;42.04;16.04;28.01]/1000;

% Heat capacity coefficients: Cp = a1 + a2*T + a3T^2 (J/mol*K),
% components arranged in same order as stoich vector

a1 = [26.63;20.04;13.39;6.25];
a2 = [0.183;0.0945;0.077;8.78*1e-3];
a3 = [-45.86;-30.95;-18.71;-0.021]*1e-6;

% Arrhenius frequency factor, k0, (1/s)
k0 = exp(34.34);
```

```
% Activation energy, Ea, (J/mol)
Ea = 284500;

% Gas constant, R, (J/mol*K)
R = 8.314;

% Reference temperature, Tref (K)
Tref = 298;

% Enthalpy of reaction at Tref, Href (J/mol*K)
H_f_298 = [-216670;-61090;-71840;0];
Href = stoich'*H_f_298;

% Mixture average thermal conductivity, kappa (J/m*K)
kappa = 2e-1;

% Mixture average diffusivity, D (m^2/s)
D = 1e-6;

% Effective heat transfer coefficient, h = UA/V where U is the overall
% heat transfer coefficient (W/m^2*K), A is reactor surface area (m^2),
% and V is the reactor volume (m^3)
h = 16500;

% Mixture average viscocity, mu (kg/m*s)
mu = 30*1e-6;


% ---------------------------------------------------------------------
% ----- SOLVER -----

% --- SOLVER PARAMETERS ---
% Number of nodes to solve for
nodes = 500;

% Error tolerance for relative residuals, generally, anything below
% 1e-6 is not recommended as it leads to oscillations in the iterations
% without further convergence
rel_tol = 1e-8;

% Max number of iterations for each time step
max_iter = 25;

% Max number of time steps
max_time_iter = 500;

% Below this tolerance, the solver will switch to a Newton algorithm,
% set to 0 if only fixed point iterations are desired. The Newton
% algorithm is efficient for problems below 10000 unknowns, but quickly
% becomes too slow for any bigger problems.
% Generally, do not set above 1e-3 since the Newton solver will converge
% to unphysical solutions without a good initial guess
newton_tol = 0;
```

```
% If true, averaging will be performed on the outputs to make the
% functions look visually smoother; introduces a small amount of error.
% However, by the mean value theorem, this error vanishes as dz -> 0.
smoothed_output = false;

% Relaxation factors, 0.6 is good for many cases with a 1e-6 rel_tol,
% but they can be any number between 0 and 1
RFCA = 0.6;           % CA relaxation factor
RFT = RFCA;           % theta relaxation factor
RFU = RFCA;           % U relaxation factor

% Time step sizes
dt = Inf;

% ---------------------------------------------------------------------
% ----- PRE-SOLVE SETUP -----

% --- CALCULATED QUANTITIES ---
rho_0 = sum(c0.*MW);             % Initial mixture density
THETA = c0/cA0;                  % Normalized initial concentrations
% Initial mixture heat capacity
K = cA0*(THETA'*a1+THETA'*a2*T0+THETA'*a3*T0^2);
p0 = sum(c0)*R*T0;               % Initial ideal gas pressure of mixture

% Calulate dimensionless numbers
Pe_M = U0*L/D;                    % Mass Peclet number
Pe_T = K*U0*L/kappa;              % Thermal Peclet number
Da = k0*L/U0;                     % First order Damkohler number
Re = rho_0*U0*L/mu;               % Reynolds number
Eu = 1/(rho_0*U0^2);              % Euler number

% Calculate recurring constants
beta = -Ea/(R*T0);                % Dimensionless activation energy
TCp_ref = stoich'*a1*Tref+(1/2)*stoich'*a2*Tref^2+(1/3)*stoich'*a3*Tref^3;

Gamma = cA0*(Href - TCp_ref)/(K*T0);  % Dimensionless reference enthalpy
N = h*L/(K*U0);                       % Number of transfer units
epsilon = -sum(stoich)*cA0/(sum(c0)*stoich(1));  % Change in moles of rxn

% Generate a uniform computational mesh
z = linspace(0,1,nodes);

% Spatial finite difference
dz = z(2) - z(1);

% Store parameters in structure for easy usage in functions
params = struct('cA0',cA0,'T0',T0,'Ta',Ta,'stoich',stoich, ...
                'rho_0',rho_0,'THETA',THETA,'K',K,'p0',p0,'MW',MW, ...
                'Pe_M',Pe_M,'Pe_T',Pe_T,'Da',Da,'Re',Re,'Eu',Eu, ...
                'beta',beta,'Gamma',Gamma,'N',N,'epsilon',epsilon, ...
                'a1',a1,'a2',a2,'a3',a3,'dz',dz,'dt',dt,'nodes',nodes);

% --- SOLUTION INITIALIZATION ---
% Maximum number of time steps
t = 1:max_time_iter;

% Initial guess for CA, assume an exponential curve
CA = (1-exp(-z))';

% Initial guess for theta, assume constant
theta = ones(nodes,1);

% Initial guess for U, assume constant
U = ones(nodes,1);

% Predict Jacobian sparsity pattern for non-linear solver
if newton_tol ~= 0
    % Main diagonals
    J1 = spdiags(ones(2*nodes,3),[-1,0,1],nodes,nodes);

    % Extra off diagonals due to backwards and forward differencing
    % at the boundaries
    J1(1,3) = 1;                % Left
    J1(nodes,nodes-2) = 1;      % Right

    jac_sparse = sparse([J1,J1,J1; J1,J1,J1; J1,J1,J1]);
    clear J1

    % Generate options structure for fsolve with Jacobian sparsity pattern
    options = optimoptions('fsolve', Algorithm='trust-region', ...
        JacobPattern=jac_sparse,FunctionTolerance=rel_tol);
end

% Count the number of linear and non-linear iterations used
Linear_Solves = 0;
Non_Linear_Solves = 0;

% Variable to store whether to use fixed point or Newton iteration.
% Start with fixed point iteration to establish convergence
newton = false;

% Preallocate residual histories
err_CA = zeros(max_time_iter,1);
err_U = zeros(max_time_iter,1);
err_T = zeros(max_time_iter,1);


% ------------------------------------------------------------------
% ----- ITERATIVE SOLVER -----
tic
for i = t
    if newton == false      % Linearized fixed point iteration

        % --- CA SOLVER ---
        % Save previous iteration to current
        CA_old = CA;
```

```
% Gather CA stencil coefficients using values from the previous
% iteration, perform a linear solve for the updated CA
[A,f] = stencil_coefficients_CA(CA,U,theta,params);
%CA = A\f;
[l,u] = ilu(A);
CA = gmres(A,f,[],rel_tol,[],l,u,CA);

% Update CA using underrelaxation
CA = (1-RFCA)*CA_old + RFCA*CA;

% --- U SOLVER ---
% Predict X, P, rho
X = 1 - U.*CA;
% Make sure X is bounded between 0 and 1, turn on if having
% convergence issues in X
% X(X<0) = 0; X(X>1) = 1;

P = pressure(X,U,theta,params);
rho = density(X,theta,P,params);
% Make sure density is positive, turn on if having convergence
% issues
% rho(rho<0) = 1e-14;

% Save previous iteration to current
U_old = U;
% Gather U stencil coefficients using values from the previous
% iteration, perform a linear solve for the updated U
[A,f] = stencil_coefficients_U(U,rho,P,params);
%U = A\f;
[l,u] = ilu(A);
U = gmres(A,f,[],rel_tol,[],l,u,U);

% Update U using underrelaxation
U = (1-RFU)*U_old + RFU*U;

% --- THETA SOLVER ---
% Update X,P field using newly calculated U
X = 1 - U.*CA;
%X(X<0) = 1e-14; X(X>1) = 1;
P = pressure(X,U,theta,params);
% Save previous iteration to current
theta_old = theta;

% Gather theta stencil coefficients using values from the previous
% iteration, perform a linear solve for the updated theta
[A,f] = stencil_coefficients_theta(X,U,P,theta,params);
%theta = A\f;
[l,u] = ilu(A);
theta = gmres(A,f,[],rel_tol,[],l,u,theta);

% Update theta using underrelaxation
theta = (1-RFT)*theta_old + RFT*theta;

% If theta drops below 0 anywhere, reset it to a small number.
```

```
        % Required to prevent the divergence of the first few iterations
        % with a poor initial guess.
        theta(theta<0) = 1e-14;

        % Increment the number of linear solves (3 per iteration)
        Linear_Solves = Linear_Solves + 3;

    else    % Non-linear Newton solver

        % Update old variables with the previous iteration
        CA_old = CA;
        U_old = U;
        theta_old = theta;

        % Generate left hand side of F(x) = 0
        F = @(y) non_linear_PFR_discretization(y,params);

        % Use the previous iteration as the initial guess for the Newton
        % solver, place into a long column vector
        x0 = vertcat(CA,U,theta);

        % Non-linear Newton solver with MATLAB's fsolve
        x = fsolve(F,x0,options);

        CA = x(1:nodes);                % Extract CA from x vector
        U = x(nodes+1:2*nodes);         % Extract U from x vector
        theta = x(2*nodes+1:3*nodes);   % Extract theta from x vector

        % Increment the number of non-linear solves
        Non_Linear_Solves = Non_Linear_Solves + 1;

        % Stop the solver if the number of non-linear solves exceeds 3
        if Non_Linear_Solves > 3
            warning(['More than 3 non-linear solves, the solution ' ...
                'may not be converging...'])
            break
        end

    end

    % Calculate L2 relative residuals
    res_CA = norm(CA-CA_old,2)/norm(CA,2)
    res_U = norm(U-U_old,2)/norm(U,2)
    res_T = norm(theta-theta_old,2)/norm(theta,2)

    % Store residual history
    err_CA(i) = res_CA; err_U(i) = res_U; err_T(i) = res_T;

    % If the residuals are below the desired tolerance, switch to the
    % non-linear Newton solver
    if res_CA < newton_tol && res_T < newton_tol
        newton = true;
    end
    % Convergence criterion: all residuals less than the relative
    % tolerance
    if res_CA < rel_tol && res_U < rel_tol && res_T < rel_tol
        break
    end

end
toc

% ----------------------------------------------------------------------
% ----- POST PROCESSING -----

% Return the correct sized residual histories
err_CA = nonzeros(err_CA);
err_U = nonzeros(err_U);
err_T = nonzeros(err_T);

% Perform Gaussian averaging on the internal nodes of the outputs,
% leaves the boundary nodes as is
if smoothed_output == true
    DD = (1/4)*spdiags([ones(nodes,1), 2*ones(nodes,1), ones(nodes,1)],...
                                    [-1,0,1],nodes,nodes);
    DD(1,:) = 0; DD(1,1) = 1; DD(nodes,:) = 0; DD(nodes,nodes) = 1;
    X = DD*X;
    U = DD*U;
    theta = DD*theta;
end

% Clear excess variables
%clear R kappa D Ea Href Pe_M Pe_T Da Re K N Ta rho_0 mu TCp_ref
%clear T0 dCp Cp Gamma F dt_CA dt_T a1 a2 a3 c0 h MW

% --- PLOT X, THETA, U, P, IN NON-DIMENSIONALIZED COORDINATES ---
figure
yyaxis left
X = 1 - U.*CA;
plot(z,X,linewidth=2, color='b')            % Plot X
ylabel('$X$',interpreter='latex')
set(gca,'YColor','b')
ylim([0,1])

yyaxis right
plot(z,theta,linewidth=2, color='r')        % Plot theta
ylabel('$\theta,\ U$',interpreter='latex')
set(gca,'YColor','k')

hold on
plot(z,U,linewidth=2, color=[1,180,40]/256,linestyle='-')   % Plot U


% Pretty plot  parameters
xlabel('$z$',interpreter='latex');
legend('$X$','$\theta$','$U$', interpreter='latex', location='best')
```

```
grid on; box on
fontname('Serif'); fontsize(16,'points');


% --- PLOT CONCENTRATION PROFILES ---
figure
c_A = cA0*(1-X)./U;
c_B = cA0*(THETA(2)-stoich(2)*X/stoich(1))./U;
c_C = cA0*(THETA(3)-stoich(3)*X/stoich(1))./U;

hold on
plot(L*z',c_A,linewidth=2,color='b')
plot(L*z',c_B,linewidth=2,color=[1,180,40]/256)
plot(L*z',c_C,linewidth=2,color='r')

% Pretty plot  parameters
xlabel('$x$ (m)',interpreter='latex');
ylabel('$c_A$, $c_B$, $c_C$ (mol/m$^3$)',interpreter='latex');
legend('$c_A$', '$c_B$', '$c_C$',interpreter='latex', location='best')

grid on; box on
fontname('Serif'); fontsize(16,'points');

% --- PLOT REACTION RATE PROFILE ---
figure
rate = k0*cA0*exp(beta./theta).*CA;

plot(L*z',rate,linewidth=2,color='b')

% Pretty plot  parameters
xlabel('$x$ (m)',interpreter='latex');
ylabel('$-r_A$ (mol/m$^3$/s)',interpreter='latex');
grid on; box on
fontname('Serif'); fontsize(16,'points');


% --- PLOT RESIDUALS ---
figure
semilogy(err_CA,linewidth=2,color='b')       % CA error

hold on
semilogy(err_T,linewidth=2,color='r')        % Theta error

semilogy(err_U,linewidth=2,color=[1,180,40]/256)    % U error

% Plot order of convergence
k = size(err_T,1);
semilogy(1:k,2.^-(1:k), linewidth=2, linestyle='--', color='k')

% Pretty plot parameters
set(gca,'YColor','k')
xlabel('Iterations, $k$',interpreter='latex')
ylabel('$r_X,\ r_\theta,\ r_U$',interpreter='latex')
legend('$r_X$', '$r_\theta$', '$r_U$', '$O(k)$', ...
    interpreter='latex', location='southwest')

grid on; box on
fontname('Serif'); fontsize(16,'points');

% ------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% Script to generate data for error convergence study of CUT solution
% algorithm.
% ------------------------------------------------------------------
% Script to solve the PFR boundary value problem using a non-linear
% finite volume scheme.
% Uses the CUT loop to solve the coupled transport problems:
%   C - Concentration equation
%   U - U momentum equation
%   T - Temperature equation
% which is iteratively solved until convergence
% ------------------------------------------------------------------

close all; clear all; clc;

% ----- INPUTS -----

% Initial concentration of A, cA0 (mol/m^3)
cA0 = 18.8;

% Initial concentrations of other species (mol/m^3)
c0 = [cA0;0;0;0];

% Initial temperature, T0 (K)
T0 = 1035;

% Heat exchanger temperature, Ta (K)
Ta = 1150;

% Reactor length, L (m)
L = 5;

% PFR velocity, U (m/s)
U0 = 2;

% Stoichiometric coefficients [a,b,c]
stoich = [-1;1;1;0];

% Molecular weights (kg/mol),
% components arranged in same order as stoich vector
MW = [58.08;42.04;16.04;28.01]/1000;

% Heat capacity coefficients: Cp = a1 + a2*T + a3T^2 (J/mol*K),
% components arranged in same order as stoich vector

a1 = [26.63;20.04;13.39;6.25];
a2 = [0.183;0.0945;0.077;8.78*1e-3];
```

```
a3 = [-45.86;-30.95;-18.71;-0.021]*1e-6;

% Arrhenius frequency factor, k0, (1/s)
k0 = exp(34.34);

% Activation energy, Ea, (J/mol)
Ea = 284500;

% Gas constant, R, (J/mol*K)
R = 8.314;

% Reference temperature, Tref (K)
Tref = 298;

% Enthalpy of reaction at Tref, Href (J/mol*K)
H_f_298 = [-216670;-61090;-71840;0];
Href = stoich'*H_f_298;

% Mixture average thermal conductivity, kappa (J/m*K)
kappa = 2e-1;

% Mixture average diffusivity, D (m^2/s)
D = 1e-6;

% Effective heat transfer coefficient, h = UA/V where U is the overall
% heat transfer coefficient (W/m^2*K), A is reactor surface area (m^2),
% and V is the reactor volume (m^3)
h = 16500;

% Mixture average viscocity, mu (kg/m*s)
mu = 30*1e-6;


% ------------------------------------------------------------------
% ----- SOLVER -----

% --- SOLVER PARAMETERS ---

% Error tolerance for relative residuals, generally, anything below
% 1e-6 is not recommended as it leads to oscillations in the iterations
% without further convergence
rel_tol = 1e-10;

% Max number of iterations for each time step
max_iter = 25;

% Max number of time steps
max_time_iter = 500;

% Below this tolerance, the solver will switch to a Newton algorithm,
% set to 0 if only fixed point iterations are desired. The Newton
% algorithm is efficient for problems below 10000 unknowns, but quickly
% becomes too slow for any bigger problems.
% Generally, do not set above 1e-3 since the Newton solver will converge
% to unphysical solutions without a good initial guess
newton_tol = 0;

% If true, averaging will be performed on the outputs to make the
% functions look visually smoother; introduces a small amount of error.
% However, by the mean value theorem, this error vanishes as dz -> 0.
smoothed_output = false;

% Time step sizes
dt = Inf;

% ---------------------------------------------------------------------
% ----- PRE-SOLVE SETUP -----

% --- CALCULATED QUANTITIES ---
rho_0 = sum(c0.*MW);              % Initial mixture density
THETA = c0/cA0;                   % Normalized initial concentrations
% Initial mixture heat capacity
K = cA0*(THETA'*a1+THETA'*a2*T0+THETA'*a3*T0^2);
p0 = sum(c0)*R*T0;                % Initial ideal gas pressure of mixture

% Calulate dimensionless numbers
Pe_M = U0*L/D;                     % Mass Peclet number
Pe_T = K*U0*L/kappa;               % Thermal Peclet number
Da = k0*L/U0;                      % First order Damkohler number
Re = rho_0*U0*L/mu;                % Reynolds number
Eu = 1/(rho_0*U0^2);               % Euler number

% Calculate recurring constants
beta = -Ea/(R*T0);                 % Dimensionless activation energy
TCp_ref = stoich'*a1*Tref+(1/2)*stoich'*a2*Tref^2+(1/3)*stoich'*a3*Tref^3;

Gamma = cA0*(Href - TCp_ref)/(K*T0);  % Dimensionless reference enthalpy
N = h*L/(K*U0);                       % Number of transfer units
epsilon = -sum(stoich)*cA0/(sum(c0)*stoich(1));  % Change in moles of rxn


% Store parameters in structure for easy usage in functions
params = struct('cA0',cA0,'T0',T0,'Ta',Ta,'stoich',stoich, ...
                'rho_0',rho_0,'THETA',THETA,'K',K,'p0',p0,'MW',MW, ...
                'Pe_M',Pe_M,'Pe_T',Pe_T,'Da',Da,'Re',Re,'Eu',Eu, ...
                'beta',beta,'Gamma',Gamma,'N',N,'epsilon',epsilon, ...
                'a1',a1,'a2',a2,'a3',a3,'dz',1,'dt',dt,'nodes',1);

% --- SOLUTION INITIALIZATION ---
% Maximum number of time steps
t = 1:max_time_iter;


% Predict Jacobian sparsity pattern for non-linear solver
if newton_tol ~= 0
    % Main diagonals
    J1 = spdiags(ones(2*nodes,3),[-1,0,1],nodes,nodes);
    % Extra off diagonals due to backwards and forward differencing
    % at the boundaries
    J1(1,3) = 1;                % Left
    J1(nodes,nodes-2) = 1;      % Right

    jac_sparse = sparse([J1,J1,J1; J1,J1,J1; J1,J1,J1]);
    clear J1

    % Generate options structure for fsolve with Jacobian sparsity pattern
    options = optimoptions('fsolve', Algorithm='trust-region', ...
        JacobPattern=jac_sparse,FunctionTolerance=rel_tol);
end

% Count the number of linear and non-linear iterations used
Linear_Solves = 0;
Non_Linear_Solves = 0;

% Variable to store whether to use fixed point or Newton iteration.
% Start with fixed point iteration to establish convergence
newton = false;



% ------------------------------------------------------------------
% ----- ITERATIVE SOLVER -----
tic

for k = 2.^(4:20)
    % Number of nodes to solve for
    nodes = k;

    % Preallocate residual histories
    err_CA = zeros(max_time_iter,1);
    err_U = zeros(max_time_iter,1);
    err_T = zeros(max_time_iter,1);

    % Generate a uniform computational mesh
    z = linspace(0,1,nodes);

    % Spatial finite difference
    dz = z(2) - z(1);

    params.dz = dz;
    params.nodes = nodes;

    % Initial guess for CA, assume an exponential curve
    CA = (1-exp(-z))';

    % Initial guess for theta, assume an exponential curve
    %theta = 0.2*(1+exp(-z))';
    theta = ones(nodes,1);

    % Initial guess for U, assume an exponential curve
    U = ones(nodes,1);
    % Relaxation factors, 0.6 is good for many cases with a 1e-6 rel_tol,
    % but they can be any number between 0 and 1
    %if nodes < 50
        RFCA = 0.3;           % CA relaxation factor
    %else
    %    RFCA = 0.6;
    %end
    RFT = RFCA;           % theta relaxation factor
    RFU = RFCA;           % U relaxation factor


for i = t
    if newton == false      % Linearized fixed point iteration

        % --- CA SOLVER ---
        % Save previous iteration to current
        CA_old = CA;

        % Gather CA stencil coefficients using values from the previous
        % iteration, perform a linear solve for the updated CA
        [A,f] = stencil_coefficients_CA(CA,U,theta,params);
        %CA = A\f;
        [l,u] = ilu(A);
        CA = gmres(A,f,[],rel_tol,[],l,u,CA);

        % Update CA using underrelaxation
        CA = (1-RFCA)*CA_old + RFCA*CA;

        % --- U SOLVER ---
        % Predict X, P, rho
        X = 1 - U.*CA;
        % Make sure X is bounded between 0 and 1, turn on if having
        % convergence issues in X
        % X(X<0) = 0; X(X>1) = 1;

        P = pressure(X,U,theta,params);
        rho = density(X,theta,P,params);
        % Make sure density is positive, turn on if having convergence
        % issues
        % rho(rho<0) = 1e-14;

        % Save previous iteration to current
        U_old = U;
        % Gather U stencil coefficients using values from the previous
        % iteration, perform a linear solve for the updated U
        [A,f] = stencil_coefficients_U(U,rho,P,params);
        %U = A\f;
        [l,u] = ilu(A);
        U = gmres(A,f,[],rel_tol,[],l,u,U);

        % Update U using underrelaxation
        U = (1-RFU)*U_old + RFU*U;
        % --- THETA SOLVER ---
        % Update X,P field using newly calculated U
        X = 1 - U.*CA;
        %X(X<0) = 1e-14; X(X>1) = 1;
        P = pressure(X,U,theta,params);
        % Save previous iteration to current
        theta_old = theta;

        % Gather theta stencil coefficients using values from the previous
        % iteration, perform a linear solve for the updated theta
        [A,f] = stencil_coefficients_theta(X,U,P,theta,params);
        %theta = A\f;
        [l,u] = ilu(A);
        theta = gmres(A,f,[],rel_tol,[],l,u,theta);

        % Update theta using underrelaxation
        theta = (1-RFT)*theta_old + RFT*theta;

        % If theta drops below 0 anywhere, reset it to a small number.
        % Required to prevent the divergence of the first few iterations
        % with a poor initial guess.
        theta(theta<0) = 1e-14;

        % Increment the number of linear solves (3 per iteration)
        Linear_Solves = Linear_Solves + 3;

    else    % Non-linear Newton solver

        % Update old variables with the previous iteration
        CA_old = CA;
        U_old = U;
        theta_old = theta;

        % Generate left hand side of F(x) = 0
        F = @(y) non_linear_PFR_discretization(y,params);

        % Use the previous iteration as the initial guess for the Newton
        % solver, place into a long column vector
        x0 = vertcat(CA,U,theta);

        % Non-linear Newton solver with MATLAB's fsolve
        x = fsolve(F,x0,options);

        CA = x(1:nodes);                    % Extract CA from x vector
        U = x(nodes+1:2*nodes);             % Extract U from x vector
        theta = x(2*nodes+1:3*nodes);       % Extract theta from x vector

        % Increment the number of non-linear solves
        Non_Linear_Solves = Non_Linear_Solves + 1;

        % Stop the solver if the number of non-linear solves exceeds 3
        if Non_Linear_Solves > 3
            warning(['More than 3 non-linear solves, the solution ' ...
                'may not be converging...'])
            break
        end

    end

    % Calculate L2 relative residuals
    res_CA = norm(CA-CA_old,2)/norm(CA,2)
    res_U = norm(U-U_old,2)/norm(U,2)
    res_T = norm(theta-theta_old,2)/norm(theta,2)

    % Store residual history
    err_CA(i) = res_CA; err_U(i) = res_U; err_T(i) = res_T;

    % If the residuals are below the desired tolerance, switch to the
    % non-linear Newton solver
    if res_CA < newton_tol && res_T < newton_tol
        newton = true;
    end

    % Convergence criterion: all residuals less than the relative
    % tolerance
    if res_CA < rel_tol && res_U < rel_tol && res_T < rel_tol
        break
    end

end
    % Save the mesh information and solution field
    X = 1 - U.* CA;
    save(['sol_' num2str(k) '.mat'], 'z','CA','U','theta','X','dz')
end
toc


% -------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% Test script plot error convergence for PFR CUT scheme. MUST run
% 'PFR_FV_test_CUT_err.m' (previous script) first to generate the
% data.
% -------------------------------------------------------------------
close all; clc;

% Mimimum/maximum exponent of the refined/coarsest mesh (h = 2^k_max)
k_max = 20;
k_min = 4;

% Load maximum refined mesh and store for comparison
temp = load(['sol_' num2str(2^k_max) '.mat'],'X','U','theta','dz');
X_star = temp.X;
U_star = temp.U;
theta_star = temp.theta;
h_star = temp.dz;

% Preallocate error vectors
```

```
e_L2_X = zeros(k_max-k_min-1,1);
e_inf_X = zeros(k_max-k_min-1,1);

e_L2_U = zeros(k_max-k_min-1,1);
e_inf_U = zeros(k_max-k_min-1,1);

e_L2_theta = zeros(k_max-k_min-1,1);
e_inf_theta = zeros(k_max-k_min-1,1);

% Loop through meshes, starting with the most refined and moving to
% the coarsest mesh
i = 1;
N = 2.^(k_max:-1:k_min);
h = zeros(k_max-k_min-1,1);

for n = N
    % Load the coarser mesh
    temp = load(['sol_' num2str(n) '.mat']);
    X = temp.X;
    U = temp.U;
    theta = temp.theta;
    dz = temp.dz;
    h(i) = dz;

    % Calculate the error by subtracting the refined mesh evaluated at
    % the coarse mesh grid points. Since the refinement halves each time
    % this means evaluating the u_star at every 'step'.
    step = 2^(i-1);
    e_X = X_star(1:step:end)-X;
    e_U = U_star(1:step:end)-U;
    e_theta = theta_star(1:step:end)-theta;

    % Calculate normalized L2 error over the domain
    e_L2_X(i) = norm(e_X)*sqrt(dz);
    % Calculate the absolute maximum error
    e_inf_X(i) = max(abs(e_X),[],'all');

    % Calculate normalized L2 error over the domain
    e_L2_U(i) = norm(e_U)*sqrt(dz);
    % Calculate the absolute maximum error
    e_inf_U(i) = max(abs(e_U),[],'all');

    % Calculate normalized L2 error over the domain
    e_L2_theta(i) = norm(e_theta)*sqrt(dz);
    % Calculate the absolute maximum error
    e_inf_theta(i) = max(abs(e_theta),[],'all');

    % Increment loop counter
    i = i + 1;

end

% Plot error on a log-log plot, compare to O(h) and O(h^2) growth
figure
```

```
% L2 error
loglog(h, e_L2_X,marker='o',color='b',linewidth=2)
hold on
loglog(h, e_L2_U,marker='o',color=[1,180,40]/256,linewidth=2)
loglog(h, e_L2_theta,marker='o',color='r',linewidth=2)

% O(h)
loglog(h,2.^-(k_max:-1:k_min),linestyle=':',color='k',linewidth=2)
% O(h^2)
f = h.^2;
loglog(h,f,linestyle='--',color='k',linewidth=2)

% Pretty plot parameters
grid on
box on
axis padded
yticks(10.^(-8:2:0))
ylim([10^-8,1])
xlabel('$\Delta z$',interpreter='latex')
ylabel('$||e||_2$',interpreter='latex')
legend('$X$', '$U$', '$\theta$', '$O(\Delta z)$', ...
        '$O(\Delta z^2)$', interpreter='latex', location='northwest')
fontname('Serif'); fontsize(16,'points');

figure
% Maximum error
loglog(h, e_inf_X,marker='o',color='b',linewidth=2)
hold on
loglog(h, e_inf_U,marker='o',color=[1,180,40]/256,linewidth=2)
loglog(h, e_inf_theta,marker='o',color='r',linewidth=2)

% O(h)
loglog(h,h,linestyle=':',color='k',linewidth=2)
% O(h^2)

loglog(h,f,linestyle='--',color='k',linewidth=2)

% Pretty plot parameters
grid on
box on
axis padded
ylim([10^-8,1])
xticks(10.^(-8:2:0))
yticks(10.^(-8:2:0))
xlabel('$\Delta z$',interpreter='latex')
ylabel('$||e||_{\infty}$',interpreter='latex')

legend('$X$', '$U$', '$\theta$', '$O(\Delta z)$', ...
        '$O(\Delta z^2)$', interpreter='latex', location='northwest')
```

Appendix C - Test Scripts for Optimal Operating Conditions

```
% ---------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 6 Apr 2025
% Sript to generate data for optimal operating conditions of acetone
% cracking in a PFR
% ---------------------------------------------------------------------
% Script to solve the PFR boundary value problem using a non-linear
% finite volume scheme.
% Uses the CUT loop to solve the coupled transport problems:
%   C - Concentration equation
%   U - U momentum equation
%   T - Temperature equation
% which is iteratively solved until convergence
% ---------------------------------------------------------------------

close all; clear variables; clc;

% cA0 and Ta values to test
c_vec = linspace(5,100,50);
T_vec = linspace(1050,1300,51);

% Preallocate conversion, ketene molar flow, and power consumption
% arrays
exit_X = zeros(length(c_vec),length(T_vec));
F = zeros(length(c_vec),length(T_vec));
W = zeros(length(c_vec),length(T_vec));

% ----- INPUTS -----

% Initial temperature, T0 (K)
T0 = 1035;

% Reactor length, L (m)
L = 5;

% PFR velocity, U (m/s)
U0 = 2;

% Stoichiometric coefficients [a,b,c,d]
stoich = [-1;1;1;0];

% Molecular weights (kg/mol),
% components arranged in same order as stoich vector
MW = [58.08;42.04;16.04;28.01]/1000;

% Heat capacity coefficients: Cp = a1 + a2*T + a3T^2 (J/mol*K),
% components arranged in same order as stoich vector

a1 = [26.63;20.04;13.39;6.25];
a2 = [0.183;0.0945;0.077;8.78*1e-3];
a3 = [-45.86;-30.95;-18.71;-0.021]*1e-6;
```

```
% Arrhenius frequency factor, k0, (1/s)
k0 = exp(34.34);

% Activation energy, Ea, (J/mol)
Ea = 284500;

% Gas constant, R, (J/mol*K)
R = 8.314;

% Reference temperature, Tref (K)
Tref = 298;

% Enthalpy of reaction at Tref, Href (J/mol*K)
H_f_298 = [-216670;-61090;-71840;0];
Href = stoich'*H_f_298;

% Mixture average thermal conductivity, kappa (J/m*K)
kappa = 2e-1;

% Mixture average diffusivity, D (m^2/s)
D = 1e-6;

% Effective heat transfer coefficient, h = UA/V where U is the overall
% heat transfer coefficient (W/m^2*K), A is reactor surface area (m^2),
% and V is the reactor volume (m^3)
h = 16500;

% Mixture average viscocity, mu (kg/m*s)
mu = 30*1e-6;

% Steam heat capacity Cp = A + BT + DT^-2
AA = 3.470;
BB = 1.450e-3;
DD = 0.121e5;
Cp = @(T) R*(AA + BB*T + DD*T.^-2);

% Boiling point of water
Tb = 373.15;


% ------------------------------------------------------------------
% ----- SOLVER -----

% --- SOLVER PARAMETERS ---
% Number of nodes to solve for
nodes = 250;

% Error tolerance for relative residuals, generally, anything below
% 1e-6 is not recommended as it leads to oscillations in the iterations
% without further convergence
rel_tol = 1e-8;

% Max number of iterations for each time step
max_iter = 25;
```

```
% Max number of time steps
max_time_iter = 500;

% Below this tolerance, the solver will switch to a Newton algorithm,
% set to 0 if only fixed point iterations are desired. The Newton
% algorithm is efficient for problems below 10000 unknowns, but quickly
% becomes too slow for any bigger problems.
% Generally, do not set above 1e-3 since the Newton solver will converge
% to unphysical solutions without a good initial guess
newton_tol = 0;

% If true, averaging will be performed on the outputs to make the
% functions look visually smoother; introduces a small amount of error.
% However, by the mean value theorem, this error vanishes as dz -> 0.
smoothed_output = false;

% Relaxation factors, 0.6 is good for many cases with a 1e-6 rel_tol,
% but they can be any number between 0 and 1
RFCA = 0.3;           % CA relaxation factor
RFT = RFCA;           % theta relaxation factor
RFU = RFCA;           % U relaxation factor

% Time step sizes
dt = Inf;

% Generate a uniform computational mesh
z = linspace(0,1,nodes);

% Spatial finite difference
dz = z(2) - z(1);

tic
for ii = 1:length(c_vec)
for jj = 1:length(T_vec)

% Initial concentration of A, cA0 (mol/m^3)
cA0 = c_vec(ii);
% Initial concentrations of other species (mol/m^3)
% i = 1        2      3        4
%     Acetone  Ketene  Methane  Nitrogen (inert)
c0 = [cA0;0;0;0];
% Heat exchanger temperature, Ta (K)
Ta = T_vec(jj);

% ---------------------------------------------------------------
% ----- PRE-SOLVE SETUP -----

% --- CALCULATED QUANTITIES ---
rho_0 = sum(c0.*MW);              % Initial mixture density
THETA = c0/cA0;                   % Normalized initial concentrations
% Initial mixture heat capacity
K = cA0*(THETA'*a1+THETA'*a2*T0+THETA'*a3*T0^2);
p0 = sum(c0)*R*T0;                % Initial ideal gas pressure of mixture
```

```
% Calulate dimensionless numbers
Pe_M = U0*L/D;                    % Mass Peclet number
Pe_T = K*U0*L/kappa;              % Thermal Peclet number
Da = k0*L/U0;                     % First order Damkohler number
Re = rho_0*U0*L/mu;               % Reynolds number
Eu = 1/(rho_0*U0^2);              % Euler number

% Calculate recurring constants
beta = -Ea/(R*T0);                % Dimensionless activation energy
TCp_ref = stoich'*a1*Tref+(1/2)*stoich'*a2*Tref^2+(1/3)*stoich'*a3*Tref^3;

Gamma = cA0*(Href - TCp_ref)/(K*T0);  % Dimensionless reference enthalpy
N = h*L/(K*U0);                       % Number of transfer units
epsilon = -sum(stoich)*cA0/(sum(c0)*stoich(1));  % Change in moles of rxn

% Store parameters in structure for easy usage in functions
params = struct('cA0',cA0,'T0',T0,'Ta',Ta,'stoich',stoich, ...
                'rho_0',rho_0,'THETA',THETA,'K',K,'p0',p0,'MW',MW, ...
                'Pe_M',Pe_M,'Pe_T',Pe_T,'Da',Da,'Re',Re,'Eu',Eu, ...
                'beta',beta,'Gamma',Gamma,'N',N,'epsilon',epsilon, ...
                'a1',a1,'a2',a2,'a3',a3,'dz',dz,'dt',dt,'nodes',nodes);

% --- SOLUTION INITIALIZATION ---
% Maximum number of time steps
t = 1:max_time_iter;

% Initial guess for CA, assume an exponential curve
CA = (1-exp(-z))';

% Initial guess for theta, assume constant
theta = ones(nodes,1);

% Initial guess for U, assume constant
U = ones(nodes,1);

% Count the number of linear and non-linear iterations used
Linear_Solves = 0;

% Preallocate residual histories
err_CA = zeros(max_time_iter,1);
err_U = zeros(max_time_iter,1);
err_T = zeros(max_time_iter,1);


% -----------------------------------------------------------------
% ----- ITERATIVE SOLVER -----
for i = t

    % --- CA SOLVER ---
    % Save previous iteration to current
    CA_old = CA;

    % Gather CA stencil coefficients using values from the
    % previous iteration, perform a linear solve for the updated
    % CA
    [A,f] = stencil_coefficients_CA(CA,U,theta,params);
    %CA = A\f;
    [l,u] = ilu(A);
    CA = gmres(A,f,[],rel_tol,[],l,u,CA);

    % Update CA using underrelaxation
    CA = (1-RFCA)*CA_old + RFCA*CA;

    % --- U SOLVER ---
    % Predict X, P, rho
    X = 1 - U.*CA;
    % Make sure X is bounded between 0 and 1, turn on if having
    % convergence issues in X
    % X(X<0) = 0; X(X>1) = 1;

    P = pressure(X,U,theta,params);
    rho = density(X,theta,P,params);
    % Make sure density is positive, turn on if having convergence
    % issues
    % rho(rho<0) = 1e-14;

    % Save previous iteration to current
    U_old = U;
    % Gather U stencil coefficients using values from the previous
    % iteration, perform a linear solve for the updated U
    [A,f] = stencil_coefficients_U(U,rho,P,params);
    %U = A\f;
    [l,u] = ilu(A);
    U = gmres(A,f,[],rel_tol,[],l,u,U);

    % Update U using underrelaxation
    U = (1-RFU)*U_old + RFU*U;

    % --- THETA SOLVER ---
    % Update X,P field using newly calculated U
    X = 1 - U.*CA;
    %X(X<0) = 1e-14; X(X>1) = 1;
    P = pressure(X,U,theta,params);
    % Save previous iteration to current
    theta_old = theta;

    % Gather theta stencil coefficients using values from the previous
    % iteration, perform a linear solve for the updated theta
    [A,f] = stencil_coefficients_theta(X,U,P,theta,params);
    %theta = A\f;
    [l,u] = ilu(A);
    theta = gmres(A,f,[],rel_tol,[],l,u,theta);

    % Update theta using underrelaxation
    theta = (1-RFT)*theta_old + RFT*theta;

    % If theta drops below 0 anywhere, reset it to a small number.
    % Required to prevent the divergence of the first few iterations
    % with a poor initial guess.
    theta(theta<0) = 1e-14;

    % Increment the number of linear solves (3 per iteration)
    Linear_Solves = Linear_Solves + 3;

    % Calculate L2 relative residuals
    res_CA = norm(CA-CA_old,2)/norm(CA,2)
    res_U = norm(U-U_old,2)/norm(U,2)
    res_T = norm(theta-theta_old,2)/norm(theta,2)

    % Store residual history
    err_CA(i) = res_CA; err_U(i) = res_U; err_T(i) = res_T;

    % Convergence criterion: all residuals less than the relative
    % tolerance
    if res_CA < rel_tol && res_U < rel_tol && res_T < rel_tol
        break
    end

end

    H_rxn = Href - TCp_ref + ....
        stoich'*a1*T0+(1/2)*stoich'*a2*T0^2+(1/3)*stoich'*a3*T0^3;
    X = 1 - U(end)*CA(end);
    FB = X*cA0*U0;
    W_tot = U0*R*T0*cA0.*log(cA0*R*T0/101325) + ...
            cA0.*(2*U0*H_rxn./((Ta-T0).*Cp(Ta)))*R.* ...
            (AA*(Ta-Tb)+0.5*BB*(Ta^2-Tb^2)-DD*(Ta^-1-Tb^-1));
    exit_X(ii,jj) = X;
    F(ii,jj) = FB;
    W(ii,jj) = W_tot;

end
end
toc

% Save cost function surface
save('PFR_optimization','exit_X','W','F','c_vec','T_vec')

% -------------------------------------------------------------------
% Patrick Heng
% 15 Feb 2025 - 3 Mar 2025
% Sript to plot ketene flow and power consumption of PFR for varying
% cA0 and Ta
% -------------------------------------------------------------------
close all; clear variables; clc

% Colormaps from '200 Colormaps' on the MATLAB file exchange,
% SWITCH THIS TO A DEFAULT COLORMAP ('parula') IF YOU DO NOT HAVE THE
% FILE INSTALLED
cmap=slanCM(100);
```

```
% Load optimization surface data
load('PFR_optimization')
% Mesh to plot over
[C,T] = meshgrid(c_vec,T_vec);

% ---------------------------------------------------------------------
% Surface plot of ketene flow for varying cA0 and Ta
figure
surf(C,T,F',linestyle='none')

% Pretty plot parameters
xlabel('$c_{A,0}$ (mol/m$^3$)',interpreter='latex')
ylabel('$T_a$ (K)',interpreter='latex')
zlabel('$F_B$ (mol/s)',interpreter='latex')
colormap(cmap)
cb = colorbar;
ylabel(cb,'$F_B$ (mol/s)',interpreter='latex')
axis tight
fontname('Serif'); fontsize(16,'points')

% ---------------------------------------------------------------------
% Surface projection plot of ketene flow for varying Ta along slices
% of constant cA0
figure
hold on
for i = 1:2:length(c_vec)
    plot(T_vec,F(i,:),linewidth=2,color= ...
                cmap(floor(256*i/length(c_vec)),:))
end

% Pretty plot parameters
xlabel('$T_a$ (K)',interpreter='latex')
ylabel('$F_B$ (mol/s)',interpreter='latex')
colormap(cmap)
cb = colorbar;
ylabel(cb,'$c_{A,0}$ (mol/m$^3$)',interpreter='latex')
clim([min(c_vec), max(c_vec)])
grid on
box on
fontname('Serif'); fontsize(16,'points')

% ---------------------------------------------------------------------
% Surface projection plot of ketene flow for varying cA0 along slices
% of constant Ta
figure
hold on
for i = 1:2:length(T_vec)
    plot(c_vec,F(:,i),linewidth=2,color= ...
                cmap(floor(256*i/length(T_vec)),:))
end

% Pretty plot parameters
xlabel('$c_{A,0}$ (mol/m$^3$)',interpreter='latex')
ylabel('$F_B$ (mol/s)',interpreter='latex')
```

```
colormap(cmap)
cb = colorbar;
ylabel(cb,'$T_a$ (K)',interpreter='latex')
clim([min(T_vec), max(T_vec)])

axis tight
grid on
box on
fontname('Serif'); fontsize(16,'points')

% ---------------------------------------------------------------------
% Surface plot of PFR power consumption for varying cA0 and Ta
figure
surf(C,T,W',linestyle='none')

% Pretty plot parameters
xlabel('$c_{A,0}$ (mol/m$^3$)',interpreter='latex')
ylabel('$T_a$ (K)',interpreter='latex')
zlabel('$\dot{W}$ (W)',interpreter='latex')
set(gca, 'ZScale', 'log')
colormap(cmap)
cb = colorbar;
ylabel(cb,'$\dot{W}$ (W)',interpreter='latex')
set(gca,'ColorScale','log')
axis tight
fontname('Serif'); fontsize(16,'points')
```