May2018

Volume 9 Issue 1

JoCsE

Journal Of Computational Science Education

Promoting the Use of Computational Science Through Education

ISSN 2153-4136 (online)

Volume 9, Issue 1 **May 2018**

JOCSE

Journal Of Computational Science Education

The Journal Of Computational Science Education (JOCSE), ISSN 2153-4136, is published quarterly in online form, with more frequent releases if submission quantity warrants, and is a supported publication of the Shodor Education Foundation Incorporated. Materials accepted by JOCSE will be hosted on the JOCSE website, and will be catalogued by the Computational Science Education Reference Desk (CSERD) for inclusion in the National Science Digital Library (NSDL).

Subscription: JOCSE is a freely available online peer-reviewed publication which can be accessed at `http://jocse.org`.

Contents

Introduction to Volume 9 Issue 1

Steven I. Gordon
Editor
Ohio Supercomputer Center
Columbus, OH
sgordon@osc.edu

Forward

In this issue, Chen presents an approach using Berkeley Madonna to help students explore chemical kinetics problems. He presents the nature of the approach used to model several different chemical systems and the exercises that were created for students. He then describes the implementation of the approach in several classes, comparing the results from using Berkeley Madonna with earlier approaches using spreadsheets and Vensim.

Hirst describes the use of systems modeling approaches used in several classes and also used with faculty who attended summer workshops. She summarizes the reactions of each of the different audiences to the use of systems models to solve problems. She also provides a number of example problems that were used in the various settings.

Ruggiero, Zhao, and Ford Versypt developed a final project assignment for an interdisciplinary applied numerical computing upper division and graduate elective related to the solution of ordinary differential equations. Students used MATLAB to build and test a graphical user interface for solving ODE's. The overall design of the assignment is reviewed along with the verification case.

Finally, Kopenhafer and O'Shea describe the results of a student modeling project testing several star formation models. Results using the modeling code Enzo to compare various approaches to modeling this system. Kopenhafer then summarizes the impact of the work on her baccalaureate program and beyond.

Teaching Kinetics through Differential Equations Constructed with a Berkeley Madonna™ Flow Chart Model

Franklin M Chen
UW-Green Bay
2420 Nicolet Drive
Green Bay, WI 54311
920-465-2834
chenf@uwgb.edu

ABSTRACT
The *alias* feature of the Berkeley Madonna platform allows this author to create a chemical kinetics project manual for students, who create flow charts solving almost all kinetics problems using rate equations. The versatile and powerful platform allows students to explore any chemical kinetics problems, from simple (e.g. 1st or 2nd order kinetics) to complex (e.g. stratosphere ozone depletion, the Lotka–Volterra mechanism), bypassing complicated syntax that is required by most of powerful mathematical programs. This kinetics manual has been successfully implemented in Physical Chemistry at UW-Green Bay in the fall semester of 2017, with the students' success rate greater than 80%.

Categories and Subject Descriptors
Physical Science and Engineering, Education

General Terms
Algorithms, Documentation, Reliability, Experimentation, Theory

Keywords
Differential equations, Educational Modules, Excel®, Berkeley Madonna™, Undergraduate, Lower or upper division of undergraduate, chemical kinetics.

1. INTRODUCTION
Although chemical kinetics is an important subject in chemistry with many potential practical applications, the subject has a lighter coverage than other topics such as equilibrium. In the second semester of general chemistry (Gen-Chem, there is only one chapter (out of 8 chapters) dedicated to kinetics.

Furthermore, when chemical kinetics is introduced to the Gen-Chem students, they were offered formulas such as first-order and 2nd-order equations which students learn through rote memory, because most of them had not learned calculus before taking the Gen-Chem classes. The upper-division undergraduate students who take "Thermodynamics and Kinetics" classes do have more opportunity to study the kinetics in more detail and use more complex equations. Those students can use calculus to derive and understand kinetic equations. Because of the resource constraint, laboratory experiments that allow students to verify their analytical solutions in kinetics are quite limited

The upper-division students may rely on computer algebra software (such as Maple or Mathcad codes) to solve more complex and interesting kinetics problems such as the Stratospheric Ozone depletion Model. Numerical codes for solving chemical kinetics are available in numerous publications [1-7]. Bentenitis [8] studied reaction mechanisms based on a stochastic simulation approach using the Chemical Kinetics Simulator (CKS) program[9]. Bigger [10] guided students to explore six fundamental kinetic processes using the ChemKinetics software.

A flow-chart based Vensim™ model was presented by Metz [11, 12] in his "Computational Chemistry" manual under cCWCS (Chemistry Collaborative Workshop for Community Scholars) in 2015. Berkeley Madonna™ [13-14] that includes a chemistry editor module has also published kinetic application examples. Applications of system-dynamics based software other than Vensim and Berkeley Madonna, such as STELLA [15], Simile [16], VisSim[17], etc have also been published, mostly in Journal of Chemical Education [18-21]. Almost all the published works cited use box and pipe diagram methods to solve kinetic problems. An *Alias-like* approach was presented by Soltzberg at 219th National American Chemical Society (ACS) meeting in 2000, and his work was cited by Metz [22] in his kinetics presentation for his Computational Chemistry for Chemical Educators (CCCE).

In this work, I select Berkeley Madonna™ as the platform[1]. The *alias*[2] feature of this platform allows students to create rate-

DOI: https://doi.org/10.22369/issn.2153-4136/9/1/1

[1] Berkeley Madonna has a free demo version that students can download and practice. Even if students choose to purchase, it is relatively low cost. The website for download is http://www.berkeleymadonna.com/index.php?route=information/static&path=bmdownloads.tpl.

equation-constructed flow charts to solve simple or complex kinetic problems. The rate equations that students write on the code are exactly the same as they learn from physical chemistry textbooks. Students can also by-pass much complicated syntax [23] to create the code, unlike other mathematical programs such as Maple™, Mathematica™, or Mathcad™. Once the code is created with the Berkeley Madonna™ platform, students can focus on the interpretations of the results generated from the code and engage in deep learning based on insights from the results.

The applications of the Berkeley Madonna™ code for pedagogical purposes is illustrated in the following examples.

2. MODULES AND PEDAGOGIES

The modules[3] consist of five initial exercises to assist students in understanding the concepts of: (1) first and 2nd order reactions, (2) equilibrium, (3) rate-determining step, (4) steady-state approximation, (5) enzyme kinetics. The first example is a tutorial helping students learn about the Berkeley Madonna code. Examples (2) to (5) are the modules for students to gain insights through the model outputs of different experimental conditions using the code. As students become more proficient in using the dynamic software, they are challenged to develop a dynamic system analyzing stratospheric ozone depletion. Each exercise is designed to allow students to build a kinetic model based on the Berkeley Madonna software, to export the parameters from the kinetic model to an Excel spreadsheet, and to explore key kinetic concepts from the Excel spreadsheet.

2.1 First order and 2nd order reaction

The models I choose to illustrate are purposely different from the conventional way of building the models. For example, for the first-order reaction, the conventional way of building the model looks like the following graph (Figure-1)

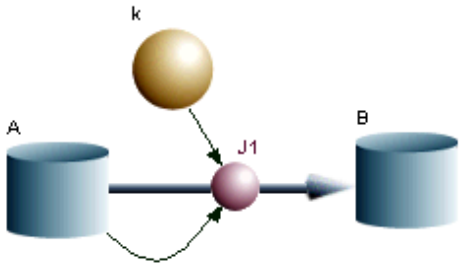


Figure-1 A conventional Berkeley Madonna model for the first-order reaction, A→B with the rate constant k.

Although the model built this way is quick and intuitive, the equation students write for *J1* is counter-intuitive:

$J1=k[A]$ [Equation 1]

instead of $-k[A]$. Most of the flow-chart based kinetic models as published [18-21] fall into this category.

In this module, students will learn to build the model as shown in Figure-2

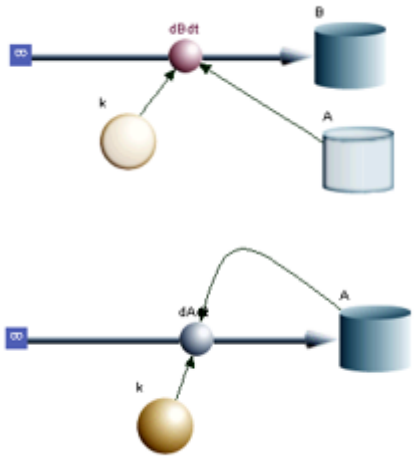


Figure-2 A tutorial Berkeley Madonna model for the first-order reaction, A→B with the rate constant k.

The new model allows students to write *d[A]/dt* and *d[B]/dt* consistently with what they learn in the classroom:

$d[A]/dt = -k[A]$; $d[B]/dt = k[A]$ [Equation 2]

After running the model, a chart of both [A] and [B] as a function of time are shown as in Figure-3.

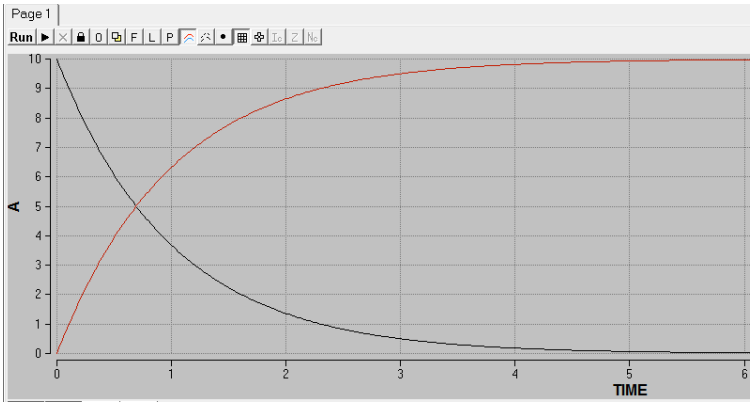


Figure-3 Concentrations of [A] and [B] as a function of time for the first order reaction. $[A]_o=10$, $[B]_o=0$, k=1.

When ln([A]) is plotted against time, a straight line is shown in Figure 4, indicative that the model built in Figure 2 is correct.

[2] It must be acknowledged that dynamic software other than Berkeley Madonna also has '*alias*' like feature such as *shadow variable* in Vensim and *ghost variable.*

[3] All reactions discussed in this manuscript refers to elementary step reactions that actually happen in the molecular level. Differential equations to represent the rate of appearance or disappearance can therefore be written as represented in each molecular elementary step reaction.

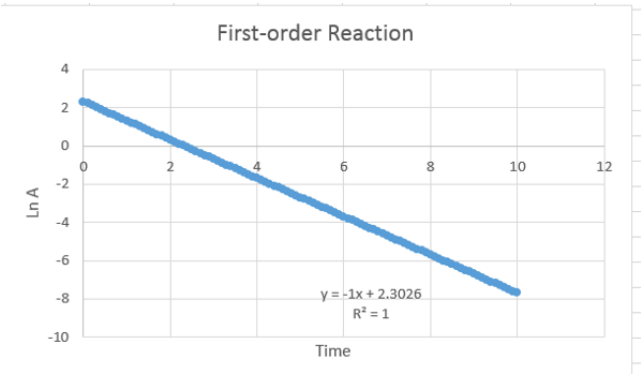


Figure-4 A plot of ln([A]) as a function of time for the 1st order reaction: A → B; A $[A]_o$=10, $[B]_o$=0, k=1. The slope is -*k*, while the intercept is *Ln([A]$_o$)*

In building the 1st order reaction model shown in Figure-2, students need to use the *alias* icon in Berkeley Madonna™. A tutorial for building a chemistry reaction model using the *alias* icon is presented in the supplementary material.

Similarly, the 2nd order kinetics can be built as shown in Figure 5:

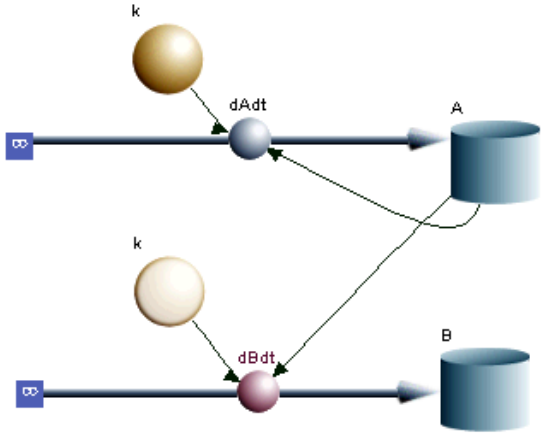


Figure-5 A tutorial Berkeley Madonna model for the 2nd order reaction, 2A→B with the rate constant k. $[A]_o$=10, $[B]_o$=0, k=1

The differential equation expression for the 2nd order reaction is:

$d[A]/dt = -2k[A]^2$; $d[B]/dt = k[A]^2$ [Equation 3]

The concentrations of both [A] and [B] as a function of time are shown in Figure-6.

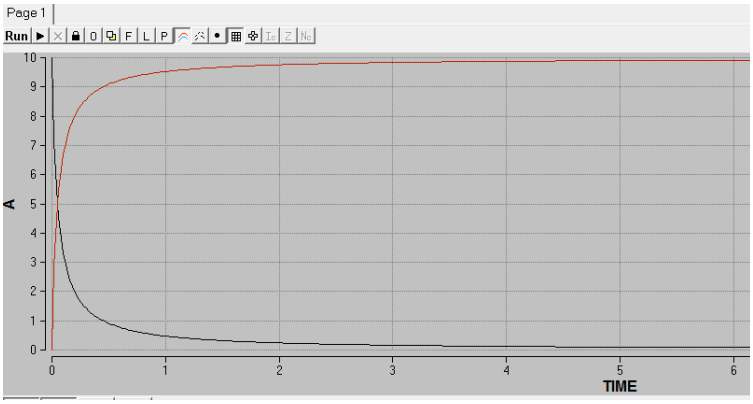


Figure-6 Concentrations of [A] and [B] as a function of time for the 2nd order reaction: 2 A → B; A $[A]_o$=10, $[B]_o$=0, k=1.

A key signature of the 2nd-order reaction is that when 1/[A] is plotted against time, a linear plot is obtained. This is indeed the case:

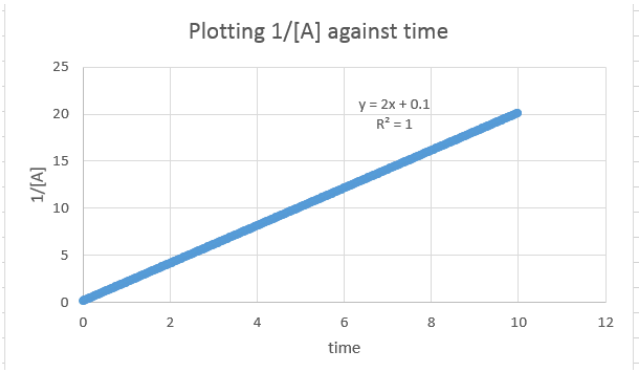


Figure-7 A plot of 1/[A] as a function of time for the 2nd order reaction: 2 A → B; A $[A]_o$=10, $[B]_o$=0, k=1. The slope is *2 k*, while the intercept is *1/[A]$_o$*

For upper-level students, the 1st and the 2nd order reactions can be used as an initial tutorial for learning Berkeley Madonna™ code. On the other hand, the spreadsheets generated from both models can be used as teaching materials for general chemistry students learning about the kinetics.

2.2 Equilibrium concept

The mathematical model to illustrate the equilibrium concept is shown in Equation [4]

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$$

$$k_1 [A] = k_2 [B]$$

$$K = \frac{[B]}{[A]} = \frac{k_1}{k_2}$$

Equation [4]

Following the tutorial presented in the first example for both the 1st or the 2nd order reactions, the finished flow chart will resemble the graph in Figure-8. The initial conditions and rate constants are: $[A]_o$= 10, $[B]_o$=0, k1=2, k2=1. The complete code is attached in the supporting materials.

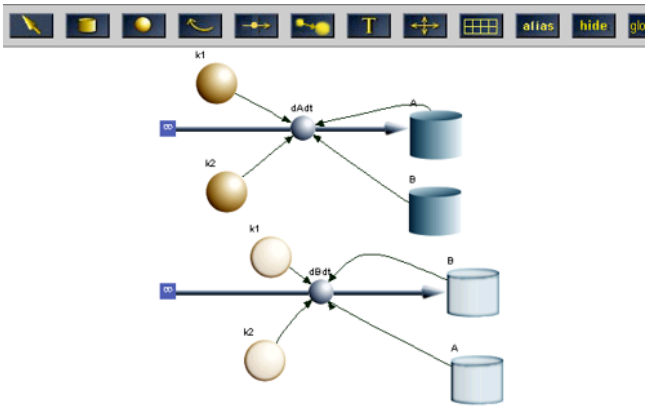


Figure-8: A flow chart to illustrate the equilibrium concept.

The differential equations for Equation [4] are:

$d[A]/dt = -k_1[A] + k_2[B]$ $\quad$ $d[B]/dt = k_1[A]-k_2[B]$ [Equation 5]

The final chart plotting both *[A]* and *[B]* as a function of time is shown in Figure-9

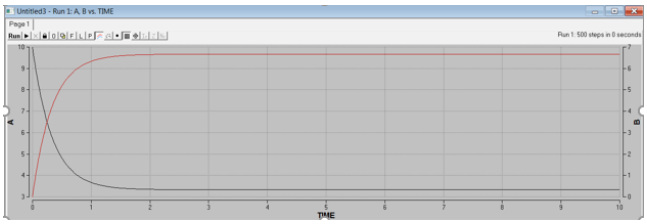


Figure-9 Concentrations of [A] and [B] as a function of time for the equilibrium reaction between A and B ; $[A]_o$=10, $[B]_o$=0, k_1=2, k_2=1

Madonna allows students to print out results in Excel format. When complete, the students are asked to answer the following questions:

[1] What time does the reaction reach equilibrium?

[2] What are the concentrations of [A] and [B] when the reaction reaches equilibrium? What is the equilibrium constant of the reaction?

The spreadsheet can also be used in general chemistry without asking students to build the flow chart model. Upper-level students can be challenged to build the model from the scratch and explain questions [1] and [2] shown above.

2.3 Rate-determining step

For a sequential reaction shown in Figure 10, the concept of the rate-determining step implies that the rate of the product formation is determined by the slowest rate constant in this sequential reaction. For example, in Figure 10, if $k_2 << k_1$, or k_3, then [D] should be determined by k_2. Mathematically, it is expressed in Eq. 6.

$$[D] \sim [A]_o (1 - e^{-k_2 t}) \quad \text{[Equation 6]}$$

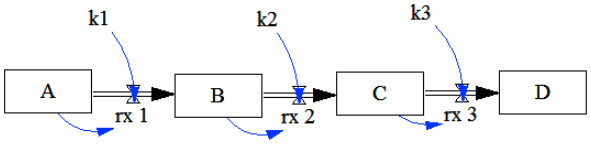


Figure-10: A flow chart to illustrate the rate-determining-step concept.

In the flow chart, the initial concentrations are $[A]_o$=100, $[B]_o$=$[C]_o$=$[D]_o$=0. Rate constants are: k_1=k_3=10, k_2=0.1

The Berkeley Madonna model when finished should look like Figure 11

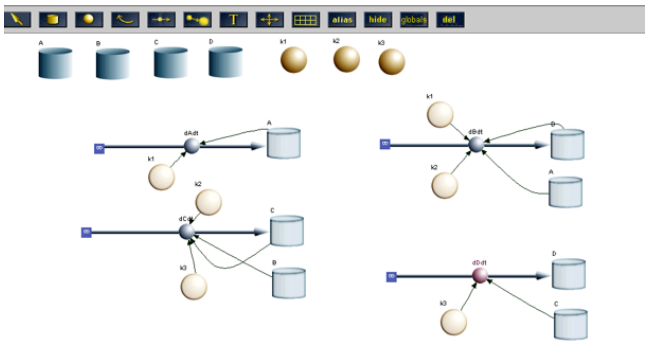


Figure-11: A complete Madonna model to illustrate the rate-determining-step concept. A→B→C→D in which B→C is the rate-determining step with $k_2=0.1$.

The differential equations that students need to write for the success of this model is:

$d[A]/dt = -k_1[A]$, $d[B]/dt = k_1[A]-k_2[B]$, $d[C]/dt = k_2[B]-k_3[C]$, $d[D]/dt = k_3[C]$ [Equation 7]

When [D] from the Excel spreadsheet (generated from the Madonna model) is compared against the plot obtained from Equation 5, they are almost identical as shown in Figure 12. Upper level students will be asked to generate the model from scratch. They are asked not only to generate the graph as shown in Figure 12, but also play with the numerical values of k_1, k_2, *and* k_3 to draw conclusions about under what kind of conditions do rate-determining step kinetics exist.

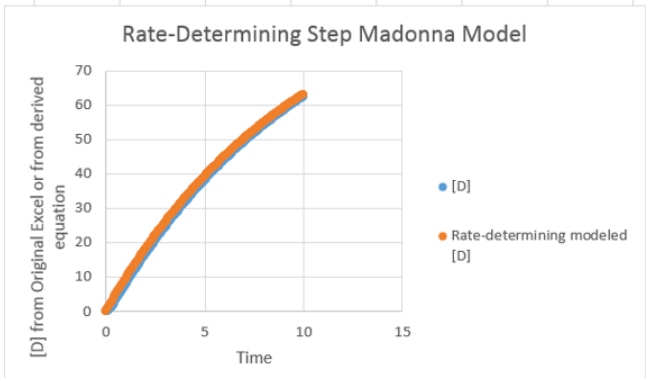


Figure-12: Excel output graph to illustrate the concentration changes of product [D] based on the Madonna output and [D] based on Equation 6.

2.4 The steady-state approximation

A flow chart to illustrate steady-state kinetics is illustrated in Figure-13, with the constraint that $k_2>>k_1$ so that $d[B]/dt = 0$.

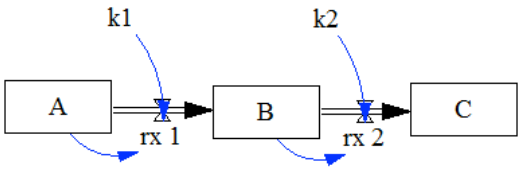


Figure-13: A flow chart to illustrate the steady-state equilibrium concept.

This implies that after an induction period in which the concentration of the intermediate, B, rises from zero, and during the major part of the reaction, the rate changes of the intermediate is negligibly small. The essential part of this exercise is that the intermediate concentration [B] should remain constant with time. A Berkley Madonna model is presented in Figure 14.

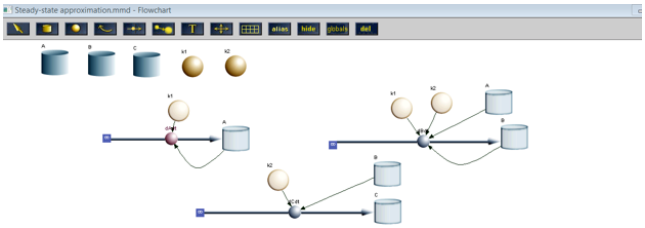


Figure-14: A complete Madonna model to illustrate the steady-state kinetics concept. A→B→C in which the rate constant k_2 for B→C is much greater than the rate constant k_1 for A→B. $[A]_o$ =100, $[B]_o$ =0, $[C]_o$ =0, k_1 =0.1, k_2 =10.

The differential equations that student will write are: $d[A]/dt = -k_1[A]$, $d[B]/dt = k_1[A] - k_2[B]$, $d[C]/dt = k_2[C]$. [Equation 8]

Figure-15 illustrates that $d[B]/dt = 0$ in which [B] is shown as the orange plot.

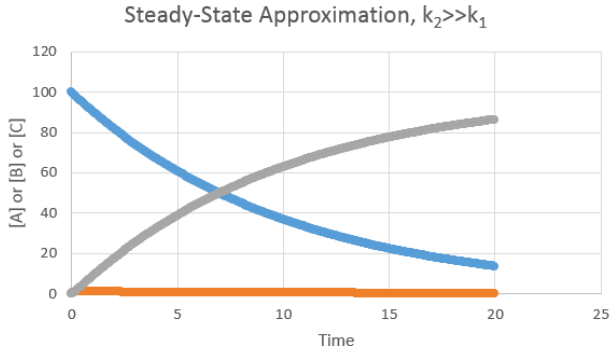


Figure-15: Excel output graph to illustrate the steady-state kinetics concept in which d[B]/dt =0 (shown orange). The reactant, [A] (blue) decreases while the product [C] (grey) increases with time.

Upper-level students are asked to reproduce this steady-state approximation from scratch. The Excel spreadsheet can be used for general chemistry students

.

2.5 The enzyme kinetics approximation and the Lineweaver-Burk Equation

A schematic description of enzyme kinetics is illustrated as follows:

$$E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} ES \xrightarrow{k_3} E + P$$

[Equation 9]

In this mechanism, E is the enzyme, S is the substrate, ES is the complex, and P is the product. In the limit where the initial substrate concentration is substantially greater than that of the enzyme ($[S]_o >> [E]_o$), the rate of product formation [24] is given by

$$R_o = \frac{d[P]}{dt} = \frac{k_3[S]_o[E]_o}{[S]_o + K_m}$$

[Equation 10]

The composite constant, K_m, in Eq.6 is referred to as the Michaelis constant of the enzyme kinetics. K_m is given by [24]

$$K_m = \frac{k_2 + k_3}{k_1}$$

[Equation 11]

Eq.10 is referred to as the Michaelis-Menten rate law. A reciprocal plot of the reaction rate can also be constructed by inverting Eq.10 which results in the Linderweaver-Burk equation,

$$\frac{1}{R_o} = \frac{1}{R_{\max}} + \frac{K_m}{R_{\max}} \frac{1}{[S]_o}$$

[Equation 12]

where $R_{MAX}=K_3 [E]_O$.

In this equation form, the plot of $1/R_o$ against $1/[S]_o$, will yield a straight line with a slope of K_m/R_{max} and an intercept of $1/R_{max}$, with $R_{max} = k_3 [E]_o$, and K_m given by Equation 11.

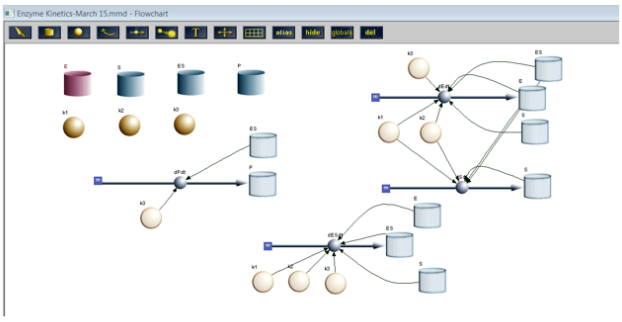


Figure-16: A Berkeley Madonna model to illustrate the enzyme kinetics mechanism. K_1=2, k_2=20, k_3=2, $[E]_o$=2.3, $[ES]_o$=0, $[P]_o$=0, $[S]_o$ varies from 2 to 20.

In this model, the differential equations expressing this kinetics are:

$d[E]/dt = -k_1 [E][S] + k_2 [ES] + k_3 [ES]$

$d[S]/dt = -k_1 [E][S] + k_2 [ES]$

$d[P]/dt = k_3 [ES]$

$d[ES]/dt = k_2 [ES] - k_3 [ES]$

[Equation 13]

The chart generated from the Excel spreadsheet is shown in Figure-17.

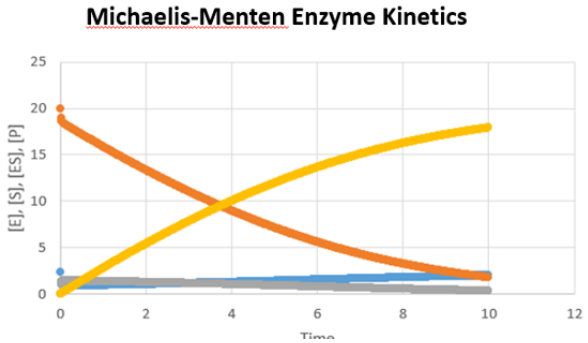


Figure-17: Excel output graph to illustrate the enzyme kinetics concept in which d[ES]/dt =0 (shown grey). Other outputs include [S] (substrate, orange), product [P] (yellow), enzyme [E] (blue)

For our Madonna model we have $R_{max} = 4.6$, $K_M = 11$. The plot of $1/R$ versus $1/[S]_o$ shown in Figure-18 is called the Lineweaver-Burk plot. A regression equation of this plot gives a slope of 1.011, and the intercept = 0.0032. These results deviate from the theoretical value in which the slope (K_M/R_{max}) is 2.3913 and the intercept ($1/R_{max}$) is 0.2174. The percent of deviation are 81% for the slope and 195% for the intercept. This deviation becomes smaller if we use a smaller initial enzyme concentration. For example, when the initial enzyme concentration is 10-fold smaller, or $[E]_o = 0.23$ instead of $[E]_o = 2.3$, then $R_{max} = 0.46$, $K_M = 11$, slope= $K_M/R_{max} = 23.913$, intercept=$1/R_{max} = 2.174$. The new Lineweaver-Burk plot yields a slope of 26.85, an intercept = 2.0342. These values only deviate from the theoretical values by 11.5% for the slope, and 6.6%. for the intercept. In the laboratory set up, usually the enzyme concentration is in the nM range while the substrate concentrations are in the mM range, so the Lineweaver-Burk equation is a useful approximation for finding the K_m values.

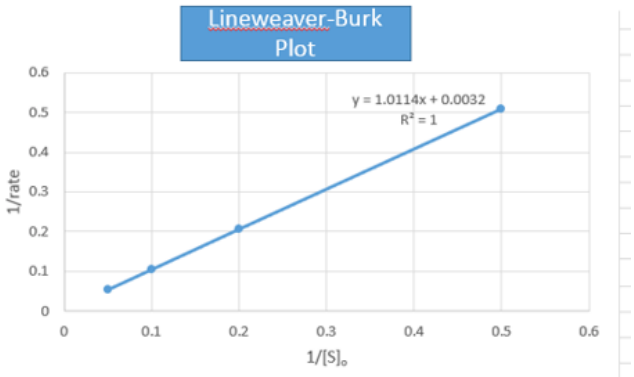


Figure-18: The Lineweaver-Burk plot of the enzyme kinetics.

2.6 Stratospheric ozone depletion kinetics model

Stratospheric (10-50 km above the sea level) ozone protects living beings from the harmful UV radiation from the sun. It was reported [25] that there was a 2.9% decline of stratosphere ozone from 1973 to 1997. The decline was due to the excess chlorofluorohydrocarbons (used as a refrigerant) released to the stratosphere. In 1995, a Nobel Prize was awarded to Professors Paul Crutzen, Mario Molina, and F. Sherwood Rowland for this important discovery.

Understanding the chemistry that controls the formation of the ozone layer, and the effects of man-made chemicals on the ozone layer, are areas in which physical chemists can impact society. Both experimental data from the laboratory and field data collected from the atmosphere are available [26]. In this unit, students learn to use Berkeley Madonna code to express both the flow-charts and differential equations for the depletion of stratosphere ozone due to chlorofluorohydrocarbons.

The elementary steps of the chemical reactions are shown as follows:

Step 1 $O_2 + hv \rightarrow 2\,O$ $k_1 = 3 \times 10^{-12}\ s^{-1}$

Step 2 $M + O + O_2 \rightarrow M + O_3$ $k_2 = 1.2 \times 10^{-33}\ cm^6\ molecule^{-2}\ s^{-1}$

Step 3 $O_3 + hv \rightarrow O + O_2$ $k_3 = 5.5 \times 10^{-4}\ s^{-1}$

Step 4 $O + O_3 \rightarrow 2\,O_2$ $k_4 = 6.9 \times 10^{-16}\ cm^3\ molecule^{-1}\ s^{-1}$

Step 5 $O + ClO \rightarrow Cl + O_2$ $k_5 = 4.12 \times 10^{-11}\ cm^3\ molecule^{-1}\ s^{-1}$

Step 6 $Cl + O_3 \rightarrow ClO + O_2$ $k_6 = 8.89 \times 10^{-12}\ cm^3\ molecule^{-1}\ s^{-1}$

[Equation 14]

Steps 1-4 are the elementary steps of the Chapman mechanism. Step-2 is the essential step of the ozone formation. Because it involves a 3-body collision of O, O_2, and an inert solid, M, the rate constant k_2 is many orders of magnitude smaller than the rate constants of other steps. The much smaller rate constant of ozone formation in the Chapman mechanism makes it susceptible to depletion in the presence of chlorine oxygen compounds as shown in steps 5-6.

The initial concentrations of O, O_2, O_3, M, ClO, and Cl taken from the same NASA resources [26] are:

$[O]_o = 1 \times 10^{7}$ *molecules/cm*^{3}

$[O_2]_o = 2 \times 10^{17}$ *molecules/cm*^{3}

$[O_3]_o = 7 \times 10^{12}$ *molecules/cm*^{3}

$[M]_o = 9 \times 10^{17}$ *molecules/cm*^{3}

$[ClO]_o = 1 \times 10^{8}$ *molecules/cm*^{3}

$[Cl]_o = 5 \times 10^{4}$ *molecules/cm*^{3}

With [Equation 14] and the initial concentrations of each species, students are asked to construct two Berkeley Madonna flow charts: One for the Chapman mechanism (Steps 1-4 in Equation 14); the other including steps 5-6 in addition to the Chapman mechanism.

The flow chart for the Chapman mechanism should look like the chart shown in Figure 19.

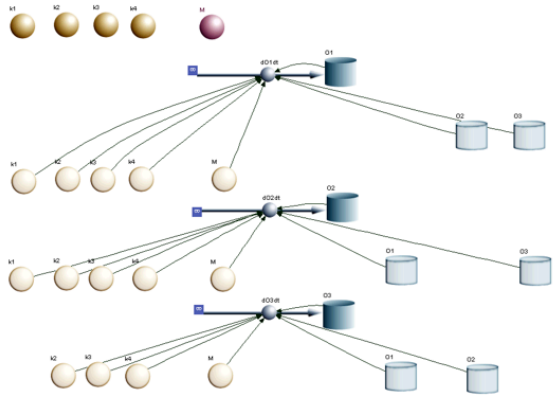


Figure-19: A Berkeley Madonna model to illustrate the Chapman mechanism. The O1, O2, and O3 shown in the chart stand for [O], [O_2], and [O_3]

The differential equations for the Chapman mechanism (Step-1 to Step-4) are:

$$d[O]/dt = 2\,k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] - k_4\,[O][O_3]$$

$$d[O_2]/dt = -k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] + 2\,k_4\,[O][O_3]$$

[Equation 15]

Figure-19: A Berkeley Madonna model to illustrate the Chapman mechanism. The O1, O2, and O3 shown in the chart stand for [O], [O_2], and [O_3]

The differential equations for the Chapman mechanism (Step-1 to Step-4) are:

$$d[O]/dt = 2\,k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] - k_4\,[O][O_3]$$

$$d[O_2]/dt = -k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] + 2\,k_4\,[O][O_3]$$

[Equation 15]

The complete mechanism is represented in the Berkeley Madonna flow chart attached as supplementary material to this manuscript.

The Excel output of $[O_3]$ in the absence and in the presence of ClO_x interference is shown in Figure 20.

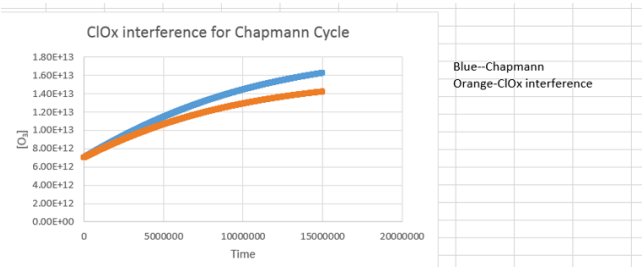


Figure-20: Ozone concentrations as a function of time in the absence and presence of ClO_x interference.

3. PROJECTS AND EXERCISES

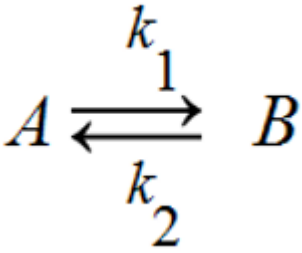
A tutorial to use differential-equation constructed flow charts with the Berkeley Madonna is presented as supplementary material to this manuscript. Although the following projects and exercises are for upper-level students, some of the Excel output can be used for general chemistry students.

3.1 First-order kinetics: A→B with rate constant k_1=1, and $[A]_o$=10

(1) Write differential equations for *d[A]/dt*, and *d[B]/dt*.
(2) Following the tutorial, construct Berkeley Madonna flow charts representing *d[A]/dt*, and *d[B]/dt.*
(3) After running the program, proceed to save the results ([A] and [B] as a function of time) in Excel© format.
(4) Plot [A] and [B] as a function of time, and discuss the results.
(5) Create another column for *ln([A])*, and plot *ln([A])* as a function of time. Discuss the results.
(6) Create another column for $[A]_o(1-e^{-k_1 t})$. Plot both *[B]* and $[A]_o(1-e^{-k_1 t})$ as a function of time and discuss the results.

3.2 Equilibrium concept

The equilibrium concept is represented as:



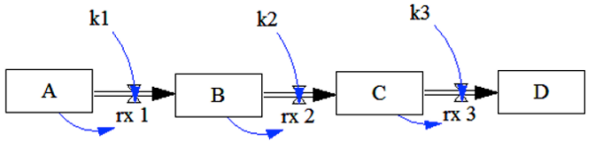
with k_1=2, and k_2=1. The initial concentrations for [A] and [B] are: $[A]_o$=10, and $[B]_o$= 0.

[1] What is an equilibrium? What is the expected equilibrium constant even before running the program?
[2] Write differential equations for *d[A]/dt* and *d[B]/dt* for this equilibrium.
[3] Construct Berkeley Madonna flow charts based on the differential equations for *d[A]/dt* and *d[B]/dt*.
[4] Run the code and save the Excel outputs. Study the Excel output and answer the following questions:

(i) Approximately when do [A] and [B] reach equilibrium?
(ii) What are the equilibrium concentrations for [A] and [B]
(iii) What is the equilibrium constant?
(iv) Show that the equilibrium constant $K = k_1/k_2$

3.3 Rate-determining step kinetics

In this exercise, students are ask to construct differential equation-based Madonna flow charts for the following diagram:



The inputs for the rate constants and initial concentrations of species are: $[A]_o$=100, $[B]_o=[C]_o=[D]_o$=0, $k_1=k_3$=10, k_2=0.1.

[1] Express in words what 'the rate-determining-step' means in chemical kinetics.

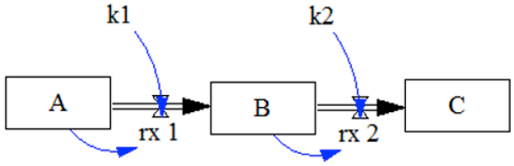
[2] Write differential equations for *d[A]/dt, d[B]/dt, d[C]/dt,* and *d[D]/dt.*

[3] Construct Berkeley Madonna flow charts based on this set of differential equations with proper inputs for k_1, k_2, k_3, $[A]_o$, $[B]_o$, $[C]_o$ and $[D]_o$.

[4] Run the code and study the Excel outputs by creating another column with the equation form, $[A]_o (1- e^{-k_2 t})$. Plot both *[D]* and $[A]_o (1- e^{-k_2 t})$ together as a function of time, and discuss the results.

3.4 The Steady-state approximation

In this exercise, students are asked to construct a differential equation based Berkeley Madonna chart diagram such that *d[B]/dt =0* in the following flow chart.



(1) Write differential equations for *d[A]/dt*, *d[B]/dt*, and *d[C]/dt*.
(2) Construct differential equation-based Berkeley Madonna chart diagrams with initial k_1=0.5, k_2=20, $[A]_o$ =10, $[B]_o$=0, $[C]_o$=0.
(3) Run the code, and save the results in Excel format. Plot *[A]*, *[B]*, and *[C]* with time. Also record *d[B]/dt* at time = 10 *s*.
(4) Change the ratio of k_1/k_2 and observe how *d[B]/dt* changes with the ratio.
(5) Change $[A]_o$ and observe how *d[B]/dt* changes with $[A]_o$.

3.5 Enzyme kinetics

In this exercise, students are asked to construct a differential equation-based Berkeley Madonna chart diagram based on the enzyme kinetics flow chart diagram shown below:

$$E + S \underset{k_2}{\overset{k_1}{\rightleftharpoons}} ES \xrightarrow{k_3} E + P$$

(1) Write differential equations for *d[E]/dt, d[S]/dt, d[ES]/dt*, **and** *d[P]/dt.*
(2) Construct differential equation based Berkeley Madonna chart diagrams with initial inputs of $[E]_o = 2.3$, $k_1 = 2$, $k_2 = 20$, $k_3 = 2$, $[S]_o = 2$.
(3) Run the code and plot *[E], [S], [P],* and *[ES]* with time.
(4) Record *d[P]/dt*, and *d[ES]/dt* at *0.1 s*.
(5) Repeat the procedures of (3) and (4), but with $[S]_o = 5$, 10, 20 then record the new *d[P]/dt* and *d[ES]/dt* at *0.1 s* for each new $[S]_o$.
(6) Construct a Lineweaver-Burk plot, obtaining the slope and the intercept of the plot, and compare the values with the theoretical values (slope = K_m/R_{max}, intercept = $1/R_{max}$, $R_{max} = k_3 [E]_o$, $K_m = ((k_2+k_3)/k_1)$. Calculate the percent of error between your experimental value (from the chart) and theoretical values (from the rate constants and the initial enzyme concentration).
(7) Repeat procedures (3), (4) (5) and (6) but with the new initial enzyme concentration of $[E]_o = 0.23$.
(8) Report your finding in words that discuss the conditions in which the Lineweaver-Burk plot applies to finding K_m in enzyme kinetics problems.

3.6 Stratospheric Ozone

In this more advanced unit, students are challenged to apply their skills to solve the stratospheric ozone depletion problems. The kinetics of stratospheric ozone consists of two parts: The Chapman mechanism (step 1-step 4) for the ozone formation, and the chloro-carbon ozone depletion (step 5-step 6). They are summarized as follows:

Step 1	$O_2 + h\nu \rightarrow 2\,O$	$k_1 = 3 \times 10^{-12}\ s^{-1}$
Step 2	$M + O + O_2 \rightarrow M + O_3$	$k_2 = 1.2 \times 10^{-33}\ cm^6\ molecule^{-2}\ s^{-1}$
Step 3	$O_3 + h\nu \rightarrow O + O_2$	$k_3 = 5.5 \times 10^{-4}\ s^{-1}$
Step 4	$O + O_3 \rightarrow 2\,O_2$	$k_4 = 6.9 \times 10^{-16}\ cm^3\ molecule^{-1}\ s^{-1}$
Step 5	$O + ClO \rightarrow Cl + O_2$	$k_5 = 4.12 \times 10^{-11}\ cm^3\ molecule^{-1}\ s^{-1}$
Step 6	$Cl + O_3 \rightarrow ClO + O_2$	$k_6 = 8.89 \times 10^{-12}\ cm^3\ molecule^{-1}\ s^{-1}$

(1) Write differential equations for *d[O]/dt, d[O2]/dt. d{O3]/dt,* for Step-1 to Step-4; the other for *d[O]/dt, d[O2]/dt. d{O3]/dt , d[ClO]/dt and d[Cl]/dt.* for Step-1 to Step-6.
(2) Construct differential equation-based Berkeley Madonna chart diagrams using the rate constants given and initial conditions for: $[O]_o = 1 \times 10^7$, $[O_2]_o = 2 \times 10^{17}$, $[O_3]_o = 7 \times 10^{12}$, $[M] = 9 \times 10^{17}$, $[ClO]_o = 1 \times 10^8$, and $[Cl]_o = 5 \times 10^4$ *molecules/cm* ^{3}.
(3) Run the code twice, once for Step-1 to Step-4; the other for Step-1 to Step- 6. The parameters used for running these two codes are: Numerical method Rosenbrock (stiff), stop-time = 1.5×10^7, $\Delta T_{min} = 500$, $\Delta T_{max} = 1000$, $\Delta T_{out} = 0$, tolerance = 0.01.
(4) Plot $[O_3]$ versus t, once for Step 1-Step 4; the other for Step-1 to Step-6
(5) Discuss the results of your plots.

4. SELECTIVE ANSWERS TO THE PROJECTS AND EXERCISES

4.1 First-order kinetics: A→B with rate constant $k_1=1$, and $[A]_o=10$

$d[A]/dt = -k_1 [A]$, $d[B]/dt = k_1 [A]$; $[A] = [A]_o e^{-k_1 t}$, $[B] = [A]_o (1 - e^{-k_1 t})$

A plot of *ln[A]* against t will give a straight line with a slope of $-k_1$. The Excel output for [B] should be the same as $[A]_o (1 - e^{-k_1 t})$.

4.2 Equilibrium concept

An equilibrium is reached when the rates of forward and backward reactions are equal. When $k_1 = 2$, and $k_2 = 1$, the forward rate is $k_1 [A]$, and the backward rate is $k_2 [B]$, $k_1 [A] = k_2 [B]$. The equilibrium constant, *K*, $K = [B]/[A] = k_1/k_2$.

4.3 Rate-determining step kinetics

When a given step is the rate-determining step, the rate constant for this specific step is many orders of magnitude smaller than the rate constants of the other steps. In this case, the rate of product formation is determined by the rate constant for this rate-determining step. Thus, if rate constant of the rate-determining step is k_2, then $[P] \sim [A]_o (1 - e^{-k_2 t})$.

4.4 The Steady-state approximation

In a mechanism of *A→B→C, B* is the intermediate. The steady-state approximation relies on the premise that *d[B]/dt =0*. To make this approximation valid, as soon as B is formed, it is immediately converted into C, or k_2 (rate constant for B→C) is many orders of magnitude larger than k_1 (rate constant for A→B). Under this circumstance, d[B]/dt ~0.

4.5 Enzyme kinetics

When differential equations are properly written and the Berkeley Madonna chart diagrams are properly constructed, a double reciprocal plot of $1/(d[P]/dt)$ versus $1/[S]_o$ will give a straight line with a slope = K_m/R_{max}, and intercept = $1/R_{max}$ in which $R_{max} = k_3 [E]_o$, $K_m = (k_2 + k_3)/k_1$. The agreement between the experimental value (from the double-reciprocal plot) and the theoretical plot (from Equation 12) will improve as the initial enzyme concentration is reduced.

4.6 Stratosphere ozone

The most important part of this exercise is to properly write the differential equations for *d[O]/dt, d[O2]/dt, and d[O3]/dt* for the Chapman mechanism (Step-1 to Step 4) given in the exercises; *d[O]/dt, d[O2]/dt, d[O3]/dt, d[ClO]/dt*, and *d[Cl]/dt* for the complete mechanism (Step-1 to Step-6).

The differential equations for the Chapman mechanism (Step-1 to Step-4) are:

$d[O]/dt = 2\,k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] - k_4\,[O][O_3]$

$d[O_2]/dt = -k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] + 2\,k_4\,[O][O_3]$

$d[O_3]/dt = k_2[M][O][O_2] - k_3[O_3] - k_4\,[O][O_3]$

When Steps 5-6 are involved, we would modify *d[O]/dt, d[O₂]/dt, and d[O₃]/dt* to include both Cl and ClO species.

$d[O]/dt = 2\,k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] - k_4\,[O][O_3] - k_5[O][ClO]$

$d[O_2]/dt = -k_1\,[O_2] - k_2\,[M][O][O_2] + k_3\,[O_3] + 2\,k_4\,[O][O_3] + k_5[O][ClO] + k_6\,[Cl][O_3]$

$d[O_3]/dt = k_2[M][O][O_2] - k_3[O_3] - k_4\,[O][O_3] - k_6\,[Cl][O_3]$

$d[ClO]/dt = -k_5[O][ClO] + k_6\,[Cl]\,[O_3]$

$d[Cl]/dt = k_5\,[O][ClO] - k_6\,[Cl]\,[O_3]$

When Berkeley Madonna chart diagrams are properly constructed and the output is saved into an Excel format, a plot of $[O_3]$ versus time with and without Cl/ClO interference will look like Figure 20.

5. TESTING AND EVALUATION

5.1 General chemistry testing and evaluations

During the spring semester of 2017, two weeks before the first exam for a general chemistry class, I provided an Excel output of first-order kinetics, $A \rightarrow B$ with rate constant k_1. I asked students to plot *ln[A]* versus *t,* using the regression equation to find the slope, and the rate constant k_1. Fifteen students out of 130 students made a mistake of taking the slope (which is a negative number) as k_1. After explanations to the class, when a similar question appeared in the first exam, only 6 out of 130 students made the same mistake.

5.2 Upper-level chemistry testing and evaluations

During the fall semester of 2016, I implemented Vensim™ projects almost exactly the same as what was presented during the 2015 cCWCS (Chemistry Collaboration and Workshop for Community Scholars) to my students of Thermodynamics and Kinetics class of 12 students. In that study, 10 out 12 students were able to completely follow the tutorial, create the diagrams and answer the questions correctly; 2 out 10 did not answer questions related to the equilibrium concept correctly even though they had created the model correctly. Even with this success, students were unable to obtain a realistic Michaelis constant, K_M through the Lineweaver-Burk plot. The class soon realized that Vensim™ was unable to model stratosphere ozone depletion problems.

The Berkeley Madonna code was implemented in the fall semester of 2017. The grading rubric is (1) Stratosphere ozone, 25/60; (2) Enzyme kinetics and Lineweaver-Weaver plot, 15/60; (3) First order kinetics, 5/60; (4) Equilibrium Concept, 5/60; (5) Rate-Determining Step, 5/60; and (6) Steady-State Approximation, 5/60. The average grade for this project was 82%. Grade distribution for this project for 26 students is shown in Figure-21.

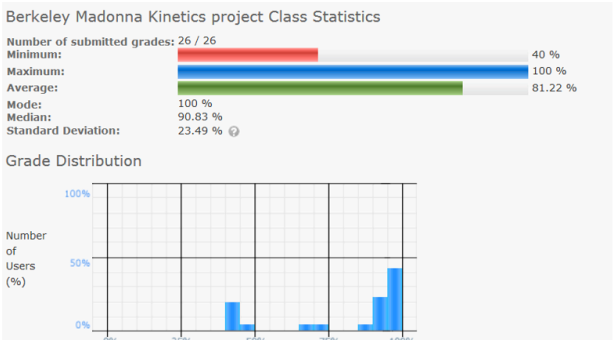


Figure 21 Grade distribution for the Berkeley Madonna Project Introduced at UW-Green Bay in Fall, 2017.

About 5 out of 26 students did not succeed in the Stratospheric Ozone problem. The common mistake was that in creating $[O_3]$ versus time in the presence of Cl and ClO species, they did not create flow charts that include *d[Cl]/dt*, and *d[ClO]/dt*. This mistake can be easily remedied by writing instructions in the manual if this project manual is introduced in 2018, or adopted by other physical chemistry instructors. Also, approximately 6 students lost points in the 1st-order kinetics plot to show a match between the product *[B]* versus time and $[A]_0\,(1\text{-}exp(-k_1t))$. If this manual is introduced again in 2018, I would add an additional assignment asking students to derive $[B(t)] = [A]_0\,(1\text{-}exp(-k_1t))$. This way, students will appreciate why this plot is included in the assignment.

At the end of the semester, the final exam (take-home exam) included a project of using the Berkeley Madonna code to create the Lotka–Volterra mechanism [27] . A successful code will create a chart similar to that shown in Figure 22. Everyone succeeded for this problem in the final exam.

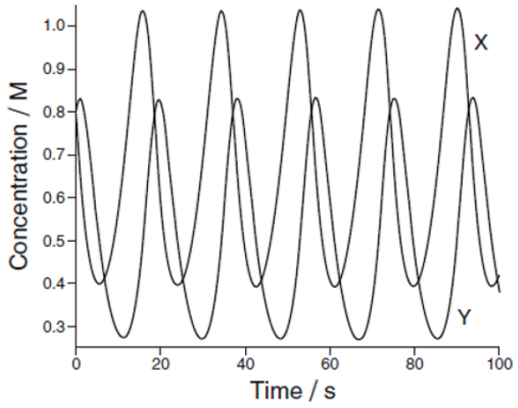


Figure-22 The oscillation pattern of the *[X]*, and *[Y]*, for the Lotka–Volterra mechanism.

6. CONCLUSION

Berkeley Madonna code was successfully adopted as a powerful and versatile platform that offers substantial pedagogical advantages for students to quickly create code and engage in interpretations. The learning outcomes for upper-level students at UW-Green Bay are encouraging. Instructors can also use the platform to create Excel spreadsheets for Gen-Chem students learning key concepts of chemical kinetics.

7. ACKNOWLEDGEMENT

Berkeley Madonna was purchased through the cCWCS (Collaborative Chemistry Workshops for Community Scholars) Implementation Grant for which this author is very grateful. He also thanks Professor Clyde Metz for introducing him to the system dynamics such as Vensim. The author thanks for the reviewers for many useful comments. He also thanks Professor Jennifer Mihalick of UW-Oshkosh for editing the writing.

REFERENCES

[1] Andraos,J., 1999. A Streamlined Approach to Solve Simple and Complex Kinetic Systems Analytically. *J. Chem. Edu.* 76(11): 1578-1583

[2] Macomber, R.S., and Constantinides, I., 1991. Modeling Complex kinetics scheme: A computational experiment *J. Chem. Edu*., 68 (12): 985-988

[3] Ferreira, M.M.C.,Ferreita, W.C.J., Lino, A.C.S.,Potto, M.E.G., 1999.Uncovering Oscillations, Complexity and Chaos in Chemical Kinetics Using Mathematica. *J. Chem Edu.,*76(6): 861-866

[4] Francl, M.M.,2004. Exploring Exotic Kinetics: An Introduction to the Use of Numerical Methods in Chemical Kinetics. *J.Chem. Edu.,* 81(10): 1535

[5] Mulquiney,P.,Kuchel,P.W. 2003. Modelling Metabolism with Mathematica; CRC Press, Boca Raton, Fl.

[6] Harvey,E.,Sweeney,R.,1999. Modeling Stratospheric Ozone Kinetics, Part I: The Chapman Cycle: OzoneModelingPart1.mcd. *J. Chem.Edu.,* 76(9): 1309

[7] Ricci, R.W., and Van Doren, J.M., 1997. Using Dynamic Software in the Physical Chemistry Laboratory. *J. Chem. Edu.,* 74(11): 1372-1374

[8] Bentenitis,N., Convenient, A., 2008.Tool for the Stochastic Simulation of Reaction Mechanism. *J. Chem. Edu*., , 85(8): 1146-1150, 2008

[9] Houle,F.A., Hinsberg,W.D., 2006. in *Real-World Kinetics via Simulations. In Annual Reports in Computational Chemistry,* Vol. 2; D.C. Spellmeyer, Ed,; Elsevier; Amsterdam

[10] Bigger,S.,2011. Chemical Kinetics: Fundamental Chemical Kinetics Principle and Analysis by Simulation. *J. Chem. Edu*., 88(2): 244

[11] Metz,C.,2015. Kinetics Tutorial based on Vensim™, cCWCS (Chemistry Collaborative Workshop for Community Scholars). Vensim™ is a free software that can be downloaded from Internet Website, last accessed 12-29-2017 http://vensim.com/free-download/.

[12] Sendinger, S.C., and Metz, C.R., 2010.Computational Chemistry for Chemistry Educators. *J. Computer Science Education* 1(1) 28

[13] Shiflet, A., Shiflet, G.W., 2017. System Dynamics Tool: *Berkeley Madonna* Tutorial 2 Internet Website, last accessed 12-29-2017

https://www.researchgate.net/publication/237270853;

[14] Krause, A., and Lowe, P.J., 2014. Visualization and Communication of Pharmacometric Models with Berkeley Madonna. CPT Pharmacometrics Syst Pharmacol, 3: e116

[15] STELLA Systems Thinking for Education and Research website last accessed 02/10/2018 https://www.dataone.org/software-tools/stella-systems-thinking-education-and-research

[16} Simile SImulistics website last accessed 02/10/2018; http://www.simulistics.com/overview.htm

[17] VisSim Simulate Multimodal Traffic with PTV Vissim in the Finest Detail. Website last accessed 02/10/2018

[18] Brooks, D.W.,1993. Technology in Chemical Education *J.Chem. Edu.,* 70(12): 991

[19] Steffen, L.K., and Holt, P.L.,1993. Computer Simulations of Chemical Kinetics, *J.Chem. Edu.,* 70(9): 705

[20] Ricci, R.W., and Van Doren, J.M.,1997. Using Dynamic Simulation Software in the Physical Chemistry Laboratory *J.Chem. Edu.,* 74(11): 1372

[21] Toby, S., and Toby, F.S.,1999. The Simulation of Dynamic Systems *J.Chem. Edu.,* 76(11): 1584

[22] Soltzberg, L.,2010. in "Computational Study of System Dynamics (Chemical Kinetics)", presented by Metz, C, Slides #34-#41 Internet Website, last accessed 2-11-2018 http://www.computationalscience.org/ccce/Lesson10/Notebook%2010%20Lecture.pdf

[23] Harvey,E., 2008. Physical Chemistry On-Line, Private Conversation

[24] Engel, T., and Reid,P.,2010. in *Thermodynamics, Statistical Thermodynamics, and Kinetics*, 2nd edition, Pearson Education Inc., Upper Saddle River, NJ, pages 497-501

[25] NASA, Earth Observatory, Internet Website, last accessed 12-29-2017, https://earthobservatory.nasa.gov/Features/Ozone/ozone.php

[26] NASA, Jet Propulsion Laboratory Internet Website, last accessed 12-29-2017 http://jpldataeval.jpl.nasa.gov

[27] J.I. Steinfeld, J.S. Francisco, and W.L. Hase 1989. , in *Chemical Kinetics and Dynamics*; Prentice Hall: Englewood Cliffs, NJ

Motivating Computational Science with Systems Modeling

Holly Hirst
Appalachian State University
Department of Mathematical Sciences
Boone, North Carolina USA 28608
HirstHP@appstate.edu

ABSTRACT

This paper describes introducing rate of change and systems modeling paradigms and software as tools to increase appreciation for computational science. A similar approach was used with three different audiences: freshman liberal arts majors, junior math education majors, and college faculty teaching introductory science courses. A description of the implementation used with each audience and their reactions to the material is discussed, along with some example problems that could be used in a variety of courses.

Keywords

Rate of Change, Systems Modeling, Computational Science

1. INTRODUCTION

The concepts of rate of change and modeling are addressed at many levels in the K-14 mathematics curriculum. Some example student learning outcomes include:

- "Interpret the rate of change and initial value of a linear function in terms of the situation it models" – Eighth Grade (Common Core Math Standards) [1]
- "Calculate and interpret the average rate of change of a function" – High School (Common Core Math Standards) [2]
- "Find a derivative interpreted as an instantaneous rate of change" – AP Calculus (ETS) [4]
- "Analyze growth and decay using absolute and relative change" – Content Learning Outcome for Quantitative Reasoning (New Mathways Project) [5]
- "Apply the mathematics they know to solve problems arising in everyday life, society, and the workplace" – High School (Common Core Math Standards) [2]
- "When making mathematical models, they know that technology can enable them to visualize the results of varying assumptions, explore consequences, and compare predictions with data" – All Levels (Common Core Math Standards) [3]
- "Apply simple mathematical methods to the solution of real-world problems" – Quantitative Reasoning for College Graduates (MAA) [6]

DOI: https://doi.org/10.22369/issn.2153-4136/9/1/2

Meeting these learning outcomes provides opportunities to introduce computation as a modeling tool to students as early as middle school. In addition, these students can be acclimated to the notion that computation is an important part of doing science, hopefully paving the way for more students to move into HPC.

In this article an outline of steps for introducing a systems modeling paradigm is presented, along with results from using software packages to investigate modeling change with three different audiences: college freshmen through a quantitative literacy course; pre-service high school teachers through a junior level mathematics modeling course; college faculty in summer workshops. In all three situations, one of the main goals was to raise the awareness of the importance of computation in doing science by modeling and solving non-trivial problems without first teaching the syntax of a standard programming language. First, consider an example modeling problem to provide some context.

2. A MOTIVATING EXAMPLE

Suppose rabbits are invading an asparagus patch, and we wish to investigate how the rabbit population affects the asparagus patch over time. What assumptions might be made? While not the only approach, we can start by thinking about what might cause increases and decreases in the number of rabbits and amount of asparagus. Some reasonable assumptions might be:

1. Rabbits are born, and how many are born depends upon the number of rabbits present to have offspring and amount of asparagus present to provide energy from food.
2. Rabbits die, and more rabbits means more competition for food, space, etc.
3. Asparagus grows steadily.
4. Asparagus is eaten by rabbits when the rabbits can find asparagus to eat.

The next step in the process is to "mathematize" these assumptions about the rates of growth and consumption of

asparagus and the rates of birth and death of the rabbits. Refining the assumptions:

1. When the number of rabbits or the amount of asparagus increase, so do the rabbit births.
2. When the number of rabbits increases, so does competition and hence rabbit deaths.
3. Asparagus grows steadily, so the growth rate is some constant amount.
4. When the number of rabbits or the amount of asparagus increase, more asparagus gets eaten.

What are some mathematical expressions that capture these ideas? A very simple proportionality argument yields the following components of the model. Let $R(t)$ represent the number of rabbits and $A(t)$ represent the amount of asparagus at a particular time t.

1. Rabbit births: proportional to both R and A implies that the increase due to births can be modeled by $r_b \times A \times R$, where the proportionality constant r_b can be interpreted as the factor controlling the rabbits' rate of reproduction.
2. Rabbit deaths: proportional to R implies a model of $r_d \times R$, where r_d can be interpreted as the fraction of the rabbit population that dies in a given time period.
3. Asparagus growth: constant a_g.
4. Asparagus consumption: $a_c \times R \times A$, where a_c can be interpreted as the fraction of interactions between rabbits and asparagus that results in a unit of asparagus being eaten.

Using these expressions we can build a mathematical formulation for our model, which can take several forms depending on the audience. Working with students who have calculus backgrounds, the model can be presented as a system of differential equations:

$$\frac{dR}{dt} = r_b AR - r_d R$$

$$\frac{dA}{dt} = a_g - a_c RA$$

$$R(0) = R_0; A(0) = A_0$$

where R_0 and A_0 represent initial quantities of rabbits and asparagus. While this approach works well for calculus-ready students, consider how to approach the problem recursively: "The quantity at a later time $(t+1)$ is the quantity now (t) ± what changed."

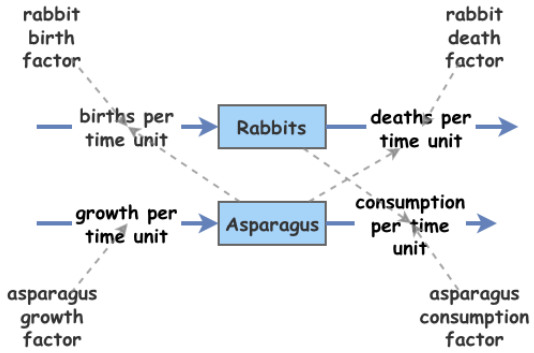
$$R(t+1) = R(t) + r_b A(t)R(t) - r_d R(t)$$

$$A(t+1) = A(t) + a_g - a_c R(t)A(t)$$

$$R(0) = R_0; A(0) = A_0$$

If the goal is spreadsheet use, these equations can be implemented a spreadsheet such as Microsoft Excel, producing a table of values documenting the changing quantities over time. Alternately, systems modeling tools allow for easy construction of the change formulas. Stella (iseesystems.com/), VensimPLE (vensim.com/), and Insightmaker (insightmaker.com/) all begin with construction of a diagram as in Figure 1.

Figure 1: Rabbit-asparagus model in Insightmaker



All of these systems modeling software environments include primitives similar to those illustrated in Figure 1: The changing quantities are represented by rectangles ("stocks"), rates of change are represented by thick arrows ("flows"), and interdependencies ("links") are represented as thin or dashed arrows. Other inputs, such as parameters, are represented by plain text or circles.

Once the diagram is completed, each primitive can be initialized for the specific model. Flows require a formula for the increase or decrease in each full time step. For example, in Figure 1 the flow labeled "consumption per time unit" that represents the rate of decrease of the asparagus contains the expression:

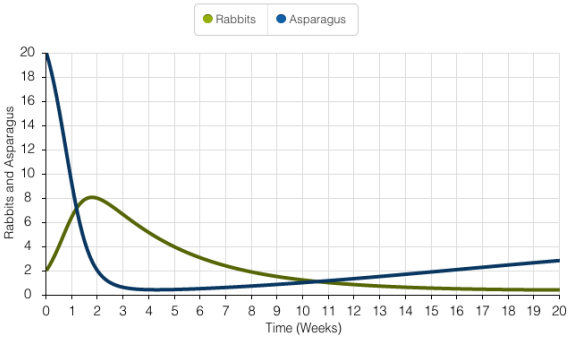
[asparagus consumption factor]*[Asparagus]*[Rabbits]

This expression is a wordier version of the proportionality description of asparagus consumption arrived at above, and all of the software packages use a "clickable list" interface to build these formulas. Stocks require initial values, which can be input as constants or as mathematical expressions involving other elements of the model. Parameters also can be modeled as constants or expressions.

After entering the relevant mathematics, the systems packages produce solutions in either tabular or graphical form. Figure 2 shows the output generated by Insightmaker for the rabbit-asparagus model with parameters: initial quantities of 2 rabbits and 20 acres of asparagus; a rabbit birth factor of 0.1; rabbit death factor of 0.3; asparagus growth factor of 0.4; asparagus consumption factor of 0.2.

One other important consideration for the systems approach to the solution is selection of the time-step. If the goal is to mimic change that happens more frequently than once per unit time (e.g., continuous change), all of the software environments allow the time-step to be set to a fraction smaller than one. In the Rabbit-Asparagus model simulation output (Fig. 2), the time-step is set to 0.125 (i.e., $1/2^3$) to approximate the continual growth of asparagus. All of the systems modeling software environments adjust the flow calculations appropriately for time-steps other than one so that growth and decay parameters do not have to be recalculated by hand.

Figure 2: Graphical solution from Insightmaker



3. IMPLEMENTING SYSTEMS MODELING IN THE CLASSROOM

The same general outline for introducing systems modeling can be used with many types of students:

1. Introduce a scenario, such as the rabbit-asparagus model. Choosing population growth allows most audiences to contribute to the discussion.

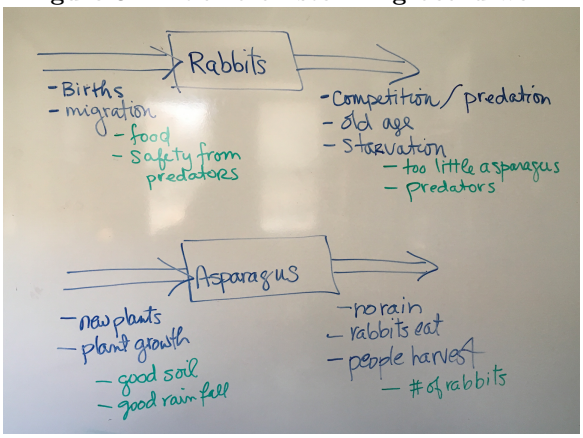
2. Draw a diagram similar to one that would be produced by the software on the board, soliciting suggestions for what might cause the populations to grow and to shrink, recording ideas on the diagram (Fig. 3).

3. Encourage the group to identify the most important components to include in the model, adding arrows to represent the interdependencies (Fig. 4).

4. Determine values for the parameters. In the absence of scientifically produced estimates for the parameters, a reasonable first attempt is to set all growth and decay parameters to be the same (e.g., 0.1). In addition, the model requires initial values for each of the populations and also a value for the time step.

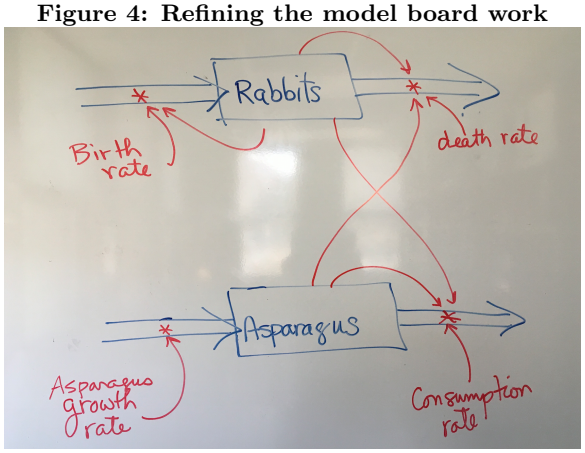
5. Build and run the model in the systems software.

Figure 3: Initial brainstorming board work



Once a working model is produced, ask students to discuss whether the model and the output makes sense. What does the software appear to be doing? What do the parameters really represent? Does it make sense that the rates are all the same? Where should parameter values really come from? Where do scientists get them? What happens to the solution if we change them? What time-step is most meaningful? Why is using a power of two as the time step important computationally?

Figure 4: Refining the model board work



Since the goal is often for the students to build their own models, it is helpful to make available a video of building the same model; a simple approach is to use screen sharing with *Hangouts on Air* (via youtube.com/live_dashboard) during the class to capture the entire conversation. That way students can view the video again anytime they need to be reminded of the intricacies of using the software.

This same approach to introducing systems modeling has been used with a variety of audiences. Sections 3.1, 3.2, and 3.3 provide detail on how systems modeling was introduced to three different audiences. Several common themes emerged from observations and discussions with these groups after the modeling activities were completed.

- Computational tools are crucial for solving problems in science.
- Mathematical formulas can be used to model real situations.
- Tools that minimize programming allow more students to experience using computation to solve problems.
- Tools that incorporate diagrams illustrating individual model components are more engaging and help students to focus on individual components of the model in the construction process.

3.1 Quantitative Literacy Course

As described in the introduction, students in a quantitative literacy (QL) course should have exposure to solving real-world problems. Systems modeling environments provide opportunities for students who are not calculus-ready to explore interesting models and gain experience with using computational tools.

3.1.1 Implementation

The approach outlined above was used in a one-week unit on modeling change in a freshman QL course, after which

the students were given a week to work in groups of three on creating and analyzing a model for the "Pollution in a Chain of Lakes" problem (see Section 5). This problem does not entail much in the way of background scientific information for the students, and so all of the students could participate in conversations about how pollution might disperse in a lake and whether the assumption of immediate mixing of pollution throughout the water in a lake would lead to unreasonable solutions.

Prior to this unit, students had significant experience with building formulas in a spreadsheet for projects in personal finance and consumer statistics; the spreadsheet was discussed as an alternate method for finding solutions to the chain of lakes problem, but students were required to use the simulation software (Stella in this particular class).

3.1.2 Reactions

During a class debriefing discussion at the completion of the project, students were asked to submit a reflection on what they learned from working on the project. Their responses indicated that they understood better how mathematical expressions could be used to model real situations and how computers are an important tool in solving problems. The students also indicated that they liked working in the visual, simulation environment more than building formulas in a spreadsheet; the process helped them to "see" the parts of the model and work on one part at a time.

3.2 Junior Math Modeling Course

The junior-level modeling course is required for preservice secondary math teachers, an especially important audience for exposure to modeling and computation given their future interactions with high school students. This course includes modeling change as a major topic in much more depth (approximately one-quarter of the course) compared to the QL course described above. The other topics covered were data modeling, linear programming, and graph / network modeling–all of which required some use of technology for solution or visualization. At this level, the students have a calculus and linear algebra background and are proficient in the use of a computer math system and a spreadsheet.

3.2.1 Implementation

As with the QL course, the discussion of modeling change began with the process outlined above for brainstorming about a scenario and creating mathematical formulas for components of the model, after which students used both a spreadsheet (Excel) and a computer math system (Maple) to implement simulations for several problems. Then students were introduced to systems modeling software (in this case Insightmaker), and asked to implement the same problems in the new environment. For the culminating project on modeling change, groups of two or three students were assigned a project from the problems in Section 4, for which they had to produce a group poster along with individually written technical reports. They were given the option of using any of the tools they wished without any attempt to influence their choice.

3.2.2 Reactions

Over the past two years, 38 of the 52 students taking the course opted to use Insightmaker, later citing the ease of use and the ability to easily incorporate more sophisticated components in their models, such as if-then, delay, and pulse functions. Similar to the QL responses, several commented on the ability to view each component of the model individually and make progressive improvements to the model components. Eight of the remaining 14 students who decided to use a different tool for their mathematical solution still drew diagrams using Insightmaker to include as part of the explanation of their model structure in their posters and reports. Course evaluations were very positive, with several of the students volunteering comments related to the modeling change portion of the course being their favorite, also indicating that they saw the relevance of solving problems computationally and thought that tools like Insightmaker could be used in their high school classrooms once they became teachers. In comparison, no such comments were made regarding spreadsheets or computer math systems, although students also volunteered positive comments on the network analysis problems and software (another diagram-based system).

3.3 Workshops for College Faculty

For two summers, the Computing MATTERS workshops (computationalscience.net), sponsored by Project XSEDE and the National Computational Science Institute were held at several universities in the Eastern and Midwestern US. During that introductory workshop, college faculty who teach entry level math and science courses were introduced to computing software, using a context of inquiry-based learning and modeling change. Most of the participants were already motivated to incorporate computation into their freshman courses, but were not sure how to work with students who may not have any computational background.

3.3.1 Implementation

In three days, the faculty were exposed to a number of computational tools, including spreadsheets (MS Excel), systems modeling (Vensim), agent modeling (Agentcubes Online), and computer math systems (Sagemath) with approximately one-half of a day spent investigating each tool. As might be expected, many levels of computational experience were represented among the participants, but the vast majority of faculty acquired enough basic knowledge of systems modeling ideas and software to begin experimenting with problems from their own fields of study. On the third day, a portion of the morning and all of the afternoon was set aside as time for the faculty to experiment more with tools or material they thought could benefit their teaching.

3.3.2 Reactions

In the flexible time on day three, the majority of faculty opted to work more on systems modeling ideas, with agent modeling a close second (both are used to model change). In discussions with the participants at the end of the workshop, it was clear that they saw the value of exposing students in introductory courses to these tools as a way to build understanding of the role computing plays in doing science–an opinion they came to the workshop with–but now many were also more confident that they could include computation without omitting material required by their course syllabi.

The NCSI follow up survey the for the workshops, administered by Project XSEDE, revealed that faculty remained excited about computational thinking and introducing com-

putational tools to their students even after the end of the workshop. While the likert scale questions on the survey did not differentiate between the different types of workshops offered, there were a number responses to the open-ended question, "[W]hat aspect of your experience in NCSI has most affected your work?", that could be clearly associated with participants from the Computing MATTERS workshops. All were positive, and several are reprinted below.

- I think about the classroom material more often in terms of systems and models, and think about how I might encourage my students to represent it that way.
- I was not aware of it before, and now see an entire new realm of teaching and research to explore and implement on our campus.
- After attending the NCSI workshop I have thought more about and have recognized the importance of developing activities and lessons that will introduce my students to Computational Thinking practices. I have used some of the software applications from the workshop with my students.
- Listening to Dr. Panoff discuss how to think about the processes of computational thinking made me concentrate on how I present problem solving in my classes.
- My own personal thought process on what constitutes a model and how to develop the means of creating the model using more than one computational tool. Excel, Agent Sheets, Interactive Physics, NetLogo are all programs I use in my classes.
- The use of software in my classes. I hace (sic) been using Vensim to explain physics concepts. Students enjoy this activity and they improve their concepts comprehension.

4. FUTURE DIRECTIONS

As indicated earlier, several common themes emerged from observations and discussions with these groups after the modeling activities were completed. In particular, gains were made in students' understanding that computation plays an important role in science and that mathematics can be used to model real situations. These results have been shared in several venues: at the 2016 NCCTM State Mathematics Conference - primarily high school teacher attendees; 2017 NCMATYC Conference - primarily community college faculty attendees; 2017 ICCS - primarily university faculty attendees. In addition, several follow up activities are planned:

- The success of the material in the QL course has resulted in plans to write a more formal module that can be used by other faculty teaching the QL course as either a short (1 week) topic or a longer (3-5 week) topic.
- The modeling tools will continue to be used in the junior modeling course, and preservice teachers who took the junior modeling course will be contacted after they begin their teaching positions to inquire about the applicability of the materials to their own classrooms.

5. EXAMPLE SCENARIOS

The following are examples of situations that have been used in all three implementations. Some are more open ended than others, but all can be addressed in a manner similar to that used for the rabbit-asparagus example.

1. **Spread of Disease**: Consider an island population and suppose that some small number of people leave the island and come back, bringing with them an infectious disease. To predict the number of persons at any time who have the disease, a simple assumption would be that the change in the number of persons who catch the disease is some fraction of the number of possible encounters between susceptible and infected people. Suppose also that the disease is one for which recovery results in immunity and it just takes some set amount of time to recover. Create a model for this situation.

   Here is a data set taken from a (mythical) island of 5000 inhabitants. Do the data support your model, i.e., are there parameters that make your model come close to the data)? How long until everyone recovers and is immune?

t (days)	0	2	6	10
sick people	5	1887	4087	3630

2. **Drug Dosage**: Clinical studies have shown that a simple model for the rate at which the concentration of a drug in the blood stream is decreasing is to assume the rate is proportional to the concentration of the drug at that time; the constant of proportionality can be thought of as the "elimination rate" for the drug. In addition, it is common for the same amount of a drug to be administered at regular intervals.

   Suppose a certain drug's elimination rate is 4% per hour, the minimum effective dosage is 0.1 mg/ml, and the maximum safe dosage is 0.3 mg/ml. Determine what size initial dose to deliver via injection and then how often a repeated dosage of 0.1 milligrams per milliliter should occur.

3. **Pollution in a Chain of Lakes**: Suppose a chain of lakes connects to each other via rivers (like the Great Lakes) and that there is a pollution source in the innermost lake. Suppose we have the following information about flow rate / volume: Lake 1 is spring fed with fresh water; 20% of lake 1 flows into lake 2 each month, 18% of lake 2 flows into lake 3 each month, and 16% of lake 3 is flushed into the ocean each month. If 100 kg/month of pollutant is dumped into Lake 1 each month for five months before the leaky pipe is found, will pollution levels ever exceed 200 kg in any of the lakes? How long before the pollution is essentially washed out to sea?

4. **Harvesting in a Shrimp Farm**: Farmed shrimp are usually raised in an enclosed area, with a known a maximum sustainable quantity of shrimp given the amount of nutrients and the size of the area. Model the rate of growth of the shrimp population assuming the change in the size of the population is proportional to the number of shrimp times a limiting factor that approaches

0 as the population of shrimp approaches the maximum sustainable population. Also incorporate plans to harvest a constant amount of shrimp each month.

Suppose we have the following parameters: maximum sustainable population = 77000; initial population = 5000. Also experience indicates that for small numbers of shrimp the population doubles each month. How does the population grow if we harvest 2000 shrimp over the course of a month? Can we harvest 5000 each month without having the population crash? What if we restock with 500 shrimplets over the course of the month?

5. **Simple Chemical Reaction**: A simple chemical reaction can be thought of as follows: A "reactant" reacts with an "intermediary" compound to produce a "product." The basic ideas behind the kinetics:

   - The rate at which the reactant changes to the intermediary during the first reaction is proportional to the amount of both the reactant and the intermediary compounds – both are needed for the reaction to occur.
   - The rate at which the intermediary then converts to the product during the second reaction is proportional to the amount of the intermediary compound that is present.
   - The two reaction rates – usually referred to as k_1 and k_2 depend on the specific compounds involved.

   Suppose 1000 moles of the reactant and 1 mole of the intermediary are added to a beaker initially, with reaction rate constants of 0.005 and 0.05 per second, respectively. How quickly does the reaction occur?

Bibliography

[1] Common Core State Standards Initiative, *Grade 8 - functions* (2016), available at http://www.corestandards.org/Math/Content/8/F/.

[2] _____, *High school : Modeling* (2016), available at http://www.corestandards.org/Math/Content/HSM/.

[3] _____, *Standards for mathematical practice* (2016), available at http://www.corestandards.org/Math/Practice/.

[4] Educational Testing Services (ETS), *AP calculus AB and AP calculus BC course and exam descriptions* (2016), available at https://www.google.com/search?q=AP+Calculus+rate+of+change+\&ie=utf-8&oe=utf-8\#q=AP+Calculus+learning+outcome+ETS.

[5] New Pathways Project, *NMP quantitative reasoning learning outcomes* (2016), available at https://www.google.com/search?q=Content+Learning+Outcome+for+Quantitative+Reasoning&ie=utf-8&oe=utf-8.

[6] Mathematical Association of America, *Quantitative reasoning for college graduates* (1994), available at http://www.maa.org/programs/faculty-and-departments/curriculum-department-guidelines-recommendations/quantitative-literacy/quantitative-reasoning-college-graduates.

Building a MATLAB Graphical User Interface to Solve Ordinary Differential Equations as a Final Project for an Interdisciplinary Elective Course on Numerical Computing

Steve M. Ruggiero*
School of Chemical Engineering
Oklahoma State University
Stillwater, OK

Jianan Zhao*
School of Chemical Engineering
Oklahoma State University
Stillwater, OK

Ashlee N. Ford Versypt†
School of Chemical Engineering
Oklahoma State University
Stillwater, OK
ashleefv@okstate.edu

ABSTRACT

A final project assignment is described for an interdisciplinary applied numerical computing upper division and graduate elective in which students develop a GUI for defining and solving a system of ordinary differential equations (ODEs) and the associated explicit algebraic equations such as values for parameters. The primary task is to use the MATLAB built-in graphical user interface development environment (GUIDE) [16, 17] to develop a graphical user interface (GUI) that takes a user-specified number of ODEs and explicit algebraic equations as input, solves the system of ODEs using `ode45`, returns the solution vector, and plots the solution vector components vs. the independent variable. The code for the GUI must be verified by showing that it returns the same results and the same figures as a system of ODEs with a known solution. The purpose of the final project assignment is threefold: (1) to practice GUI design and construction in MATLAB, (2) to verify code implementation, and (3) to review content covered throughout the course. The manuscript first introduces the course and the context and motivation for the project. Then the project assignment is detailed. Two student project submissions are described. The verification case study is also provided.

KEYWORDS

numerical methods, graduate education, computational science elective course

1 INTRODUCTION

The corresponding author of this paper teaches a course titled Applied Numerical Computing for Scientists and Engineers that she developed at Oklahoma State University. The course is offered as a chemical engineering upper division and graduate elective designed and advertised to be interdisciplinary. Over the first two offerings, five chemical engineering seniors took the course along with the following numbers of graduate students by degree program: eight in chemical engineering, six in mechanical engineering, two in chemistry, and one each in environmental engineering, mathematics, and plant and soil sciences. The first and second authors of this paper took the course as first year chemical engineering graduate students in the first and second course offerings, respectively.

The course is designed to train science and engineering seniors and graduate students to use practical software tools for computational problem solving and research: Git for version control, LaTeX for mathematical and scientific typesetting, and MATLAB and Python as high level programming languages with libraries of solvers, visualization tools, and capabilities for designing graphical user interfaces (GUIs). Throughout the course, the instructor emphasizes best practices of open-source code development, verification, and documentation [2, 3, 8–15, 18, 20–23, 26, 28] and adopts the Software Carpentry philosophy of providing hands-on training in very practical computational skills for scientists and engineers [27]. Rather than covering the basics of computer programming or details of algorithms for numerical methods, the course focuses on applying numerical computing methodologies, primarily ordinary differential equation solvers and optimization routines for parameter estimation to solve realistic continuum scale problems in science and engineering.

The course consists of ten reading assignments worth 2% each and six computational assignments. The first assignment is worth 5% and introduces the students to using version control in Git and document typesetting in LaTeX, which are required components in all subsequent assignments. The second assignment is worth 5% and gives student practice with programming in MATLAB while developing best practices for scientific computing. The students are required to create a function defining a system of ordinary differential equations (ODEs) and write well-documented code. The third assignment is worth 10% and involves using built-in functions and library routines for numerical methods (specifically ODE solvers) in MATLAB and Python to solve the system of ODEs described in Appendix A. The fourth assignment is worth 15% and covers parameter estimation of dynamic models using MATLAB and Python focusing on a case study from [1]. The fifth assignment is worth 15% and involves creating a relatively straightforward GUI in MATLAB to take user inputs and display simulation results from user-defined functions provided by the instructor for different scientific applications. The sixth and final computational assignment is worth 30% of the course grade and is described in detail in this paper.

*The first two authors contributed equally to this work.
†Corresponding author.

DOI: https://doi.org/10.22369/issn.2153-4136/9/1/4

The final computational assignment is referred to as the "final project" to emphasize its significance in the course grade, its comprehensive nature, and the time involved to complete the assignment satisfactorily. The students are allowed one month to work on this project, and it serves as a take-home final exam. The purpose of this assignment is threefold: (1) to practice GUI design and construction in MATLAB, (2) to verify code implementation, and (3) to review content covered throughout the course. The project builds upon content from all of the previous assignments. Although it does not explicitly involve parameter estimation, the systems of ODEs used for the dynamic models in the fourth assignment from [1] are used in the final project as two of the three verification cases required. For brevity we only provide the most complicated of the verification cases here in Appendix A, which is from the third assignment.

In the chemical reaction engineering course that the corresponding author teaches and many other offerings of the same course around the U.S., the software POLYMATH [25] is used as recommended by the popular textbook [4], which uses the software throughout the book in many examples and homework problems. POLYMATH is a numerical computation package that can solve data regressions and systems of simultaneous ODEs, linear algebraic equations, or nonlinear algebraic equations. These are all of the types of computational science problems encountered in a typical chemical reaction engineering course. In POLYMATH, equations are entered through GUIs in formats that look just like the written form of the equations, so there is not a coding learning curve (Fig. 1 and Fig. 2). After entering equations via GUIs or in lieu of using the GUIs, users can edit code directly in a script file. For details on how to use POLYMATH to enter and solve a system of ODEs, a tutorial document is available on the web [5]. A limitation of the current POLYMATH educational version is that users can enter only 30 simultaneous ODEs and 40 explicit algebraic equations. This is adequate for typical use. However, for the course project in the author's chemical reaction engineering course [6], the limit is often reached. Additionally, POLYMATH cannot combine different types of numerical solvers in the same problem unlike other software programs such as MATLAB and Python. The author originally started programming MATLAB or Python scripts for working around the equation number limit in POLYMATH but found the codes to be challenging for novice users to modify. The final project discussed here grew out of a need to have a POLYMATH-like GUI-based platform built into a more sophisticated software platform when dealing with more than 30 simultaneous ODEs. MATLAB was selected for this final project because of its relative ease in creating GUIs and packaging them as apps, which are very user-friendly for download and installation.

The remainder of this paper provides the required tasks in the final project assignment in Section 2, gives details of two student project submissions in Section 3, and offers conclusions in Section 4. The verification case study is presented in Appendix A.

2 PROJECT ASSIGNMENT

In the project assignment, students develop a GUI for defining and solving a system of ODEs and the associated explicit algebraic equations (e.g., temperature dependence of rate constants or values for parameters) for different applications. The primary task is to use the MATLAB built-in graphical user interface development environment (GUIDE) to develop a graphical user interface (GUI) that takes a user-specified number of differential equations and explicit algebraic equations as input, solves the system of ODEs using `ode45`, returns the solution vector, and plots the solution vector components vs. the independent variable. The GUI should work in general for any system of ODEs (specifically initial value problems not boundary value problems). The code for the GUI must be verified by showing that it returns the same results and the same figures as the systems of ODEs defined in Computational Assignments 3 and 4. The more complicated of these verification cases is provided in detail in Appendix A. Students are not penalized if their code does not work for arbitrary ODEs beyond the verification cases.

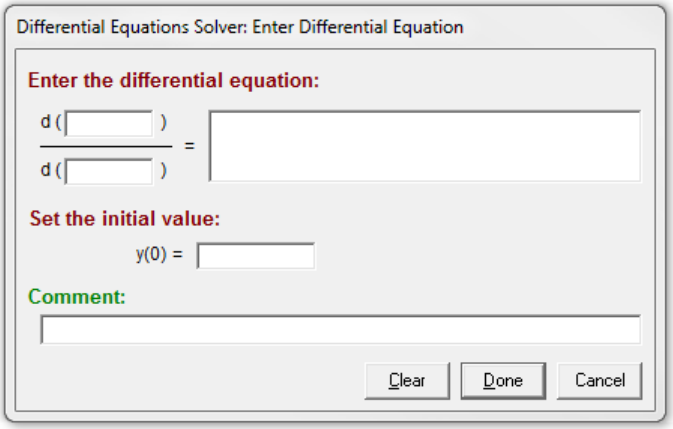


Figure 1: Graphical user interface in POLYMATH for entering a differential equation.

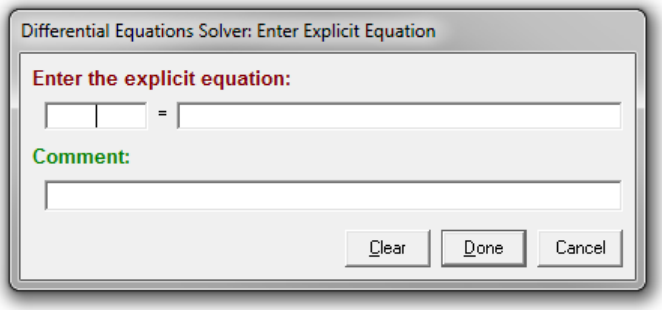


Figure 2: Graphical user interface in POLYMATH for entering an explicit algebraic equation.

The GUI can take a variety of layouts, which may involve use of multiple `.m` files, if desired. A descriptive file naming system should be used. Students should design a GUI that is intuitive to use with the following required as buttons:

- define a system of ODEs
  - The numbers of differential equations and algebraic equations should be solicited either directly or indirectly.

    - In the direct method, when this button is clicked, a window should appear asking the user how many equations of both types that they want to define. After that windows should appear recursively for the user to enter the specified numbers of corresponding differential equations and algebraic equations.
    - In the indirect method, when this button is clicked, windows should be opened for defining differential or algebraic equations one at a time (or in bulk for algebraic equations). At the end of each input, the user chooses done or continue to enter another equation of the same type. The number of equations of that type is then incremented.
  - The interfaces for entering the equations should be similar to those used in POLYMATH (Figs. 1 and 2), which can be implemented using one or more windows that are themselves GUIs and/or dialog boxes.
  - The user must be able to enter initial values and limits of integration.
  - The comment section and the buttons labeled *Clear* and *Cancel* shown in the POLYMATH examples (Figs. 1 and 2) are optional.
  - After input is provided by the user, it should be visible to them as a static textbox, editable textbox, or listbox with options to edit the input.
- calculate results
  - This button should calculate results in preparation for either exporting to Excel or plotting.
  - Something should be output to indicate that the results were calculated; command window output is acceptable.
- plot results
  The plot routine should have options for displaying all or a subset of the output vectors (dependent variables vs. the independent variable).
- save plot
  - The save plot button should call `export_fig.m` [19] to create a .png file of a reasonable size for the output plot that currently appears on the screen.
  - The user should be prompted to specify a filename and target directory for the plot.
  - A sample callback function for a button that saves a plot with a prescribed name via `export_fig.m` is shown in Fig. 3.
- export results as Excel file
  The export results button should export all the independent and dependent variable output from the ODE solver to a `.csv` or `.xls` or `.xlsx` file (label the type of output on your GUI) so that the user could view the results in Excel for creating more elaborate plots.

Additional buttons or other GUI objects may be useful. For example, a `.mat` file is recommended for saving selected variables from a subsidiary GUI window for defining the ODEs or the explicit equations. Then this can be loaded in the main GUI file to read the values after the temporary GUI window closes. A cell array is a natural way to save the values generated from each of the GUI windows; however, other ways are acceptable. Another useful button could allow users to edit the axes labels, particularly to include units. The GUI must be packaged as a MATLAB app and submitted electronically via the course website.

```
% Executes on button press in saveplot_button
function saveplot_button_Callback(hObject, ...
    eventdata, handles)
% hObject handle to saveplot_button
% eventdata reserved
% handles structure with handles and user data
if exist('plotName.png', 'file') == 2
    beep
    h = msgbox(...
    'plotName.png already exists. Please rename ...
        or delete the existing plotName.png file ...
        before trying to save again.',...
    'Plot NOT Saved');
else
    ax = handles.plotAxes;
    figure_handle = isolate_axes(ax);
    export_fig plotName.png
    h = msgbox(...
    'The plot was successfully saved as ...
        plotName.png. Be careful to rename it if ...
        you want to save multiple versions of ...
        the plot.',...
    'Plot Saved');
end
```

Figure 3: `saveplot_button_Callback` function example.

Students must create a `.tex` file to document testing that their GUI works for the verification test cases. A screenshot of the GUI when all equations have been entered must be included as well as the output plot with all of the dependent variables plotted together vs. the independent variable. A verification case is described further in Appendix A.

String manipulation is not the primary purpose of the final project, but it is necessary to take input from different windows and compose strings into equations. To aid students less familiar with string manipulation, the following tips are provided. If the code can accept the proper input arranged as cell arrays, the function shown in Fig. 4 connects the input to the ODE solver. In test cases, it is clear that the numerator and denominator are not exactly the strings that are needed for `ode45(@(t,y)ODE...)` and that the explicit equations and the right hand sides of the ODEs are not in terms of y and t as desired. It is strongly advised NOT to use `strrep` in MATLAB or other find and replace algorithms. Instead, let MATLAB do that automatically by parsing the strings. Fig. 4 is a working version of a function `ODE` that properly reads cell array inputs stored as variables in `handles` and converts them to the equations that define the system of ODEs. `ODE` is called by `ode45`. The only requirement to the user is that they cannot name a constant parameter y or t. This requirement is not necessary for independent or dependent variables.

3 STUDENT PROJECT SUBMISSIONS

The top student submissions for the final project from each of the first two course offerings are presented here as examples. We have prepared a private Bitbucket version control repository for

```
function dydt = ODE(t, y, denominators, ...
     numerators, RHSs, explicitEqns)
% Input: t and y are the independent and
%   dependent variable values,
%   denominators, numerators, RHSs, and
%   explicitEqns are cell arrays with
%   the first three terms defining the
%   ODEs of the formă
%   d(numerators) / d(denominators) = RHSs
%   and the fourth term defining the associated
%   explicit equations
% Output: the derivative vector dy/dt for
%   y(1):y(numODEs) where
%   numODEs = length(numerators) = length(RHSs) =
    length(denominators)
% To be called by ode45(@(t,y)...
%   ODE(t,y,denominators,numerators,RHSs,...
%   explicitEqns), tspan, y0)

% independent variable
str0=cell2mat(denominators(1));
eval(strcat(str0,'=t;'));
% dependent variables
for i = 1:length(numerators)
    str1 = cell2mat(numerators(i));
    eval(strcat(str1,'=y(i);'));
end
% explicit equations provided in MATLAB
% acceptable order; can be a
% semicolon separated list
for i = 1:length(explicitEqns)
    str2 = cell2mat(explicitEqns(i));
    eval(str2);
end
% Right-hand sides of ODE definitions
for i = 1:length(RHSs)
    str3 = cell2mat(RHSs(i));
    dydt(i) = eval(str3);
end
% output formatting
dydt = dydt';
```

Figure 4: Example `ODE` function that properly reads cell array inputs stored as variables in handles and converts them to the equations that define the system of ODEs.

archiving the code for these submissions along with the instructor documents related to the project assignment. The link is not shared publicly here to prevent future students from simply downloading the solution without doing the project. Interested educators may contact the corresponding author by email to request access to the private repository.

3.1 Submission 1

3.1.1 Overview. The program of Submission 1 is mainly composed of three parts: collecting user inputs to define the system of ODEs and the explicit equations, solving the system of ODEs coupled to the explicit equations, and saving simulation results by exporting calculated values and figures of plots.

3.1.2 GUI Description. The main GUI is composed of ten push buttons, three listboxes, and one axes (plot) object (Fig. 5). The user can provide inputs through the push buttons. List boxes are used to display the ODEs, corresponding initial conditions, and the explicit equations. The axes object is used to display plots of some or all of the dependent variables as functions of the independent variable (Fig. 5 shows only the first dependent variable selected). Multiple GUI windows are used for this program.

All the push buttons, except `Reset` and `Help`, were separated into three groups based on the objectives of the program. Within the panel labeled `Define/Edit Equations/Parameters`, the `Define Equations` push button first allows the user to specify the number of ODEs and gives the user an option to choose whether or not to define any explicit equations after the system of ODEs is defined (Fig. 6). If the radio button `Add explicit equations` is activated (Fig. 6), a GUI window appears that allows user to enter the explicit equations in bulk (Fig. 7) after the system of ODEs has been successfully defined. All the ODEs must have explicit form of $\frac{dy}{dt} = f(t, y)$. Based on the input number of ODEs, a `for` loop is used to repeatedly pop up the window for defining each ODE separately along with its initial value (Fig. 8). When the loop is finished, a new GUI window lets the user define the upper and lower limits of the integration, with both having default values of 0 (Fig. 9). The specified values and equations all appear in the main GUI window either in listboxes or as static text (Fig. 5).

The remaining buttons in the first panel in Fig. 5 are used for editing. The `Edit Selected ODE` push button enables the user to edit specific ODE expressions and the corresponding initial value according to selected line in the `ODE(s) Entered` list box (Fig. 5). By clicking the `Integration Limits` push button, the user can modify the integration limits via the dialog box (Fig. 9). Through the `Edit Explicit Eqs` push button, the user opens a new GUI window (Fig. 7) to define the explicit equations if they have not been defined yet or to edit existing explicit equations.

In the `Calculate/Plot` panel, clicking the push button `Calculate` opens a dialog window to require the user to enter the step size for the numerical integration. The default value is 10^{-4}. Then `ode45` is called to solve the system of ODEs. When the `Plot Results` button is pushed, the window titled `Select Variables Shown on Plot` appears to allow the user to select single or multiple dependent variables to be shown on the axes of the main GUI (Fig. 10). The user can modify the labels of the independent and dependent axes via the push button `Edit Plot Axes`. The default axes labels are shown in Fig. 11.

The `Save Plot` push button enables the user to specify a file name and target local directory for saving the plot currently shown in the axes area of the main GUI. Similarly, the `Save Data to .csv` push button prompts the user to provide a file name and local directory for saving all the results for the independent and dependent variable output from the ODE solver to a `.csv` file.

The program is capable to some extent of checking for the legality of user inputs. In the window to define a differential equation (Fig. 8), the numerator and denominator positions are checked for the presence of characters and the initial condition blank is checked for a numerical value. The upper limit of integration is required to be larger than the lower limit. When the user decides to edit the notation of the independent variable, the program can only change the left hand side of every ODE; therefore, the user needs to make sure the notation is also modified on the right hand side of each ODE to avoid any errors during the calculation.

3.1.3 Program Verification. We entered the system of equations from Appendix A into the program, calculated the results, and

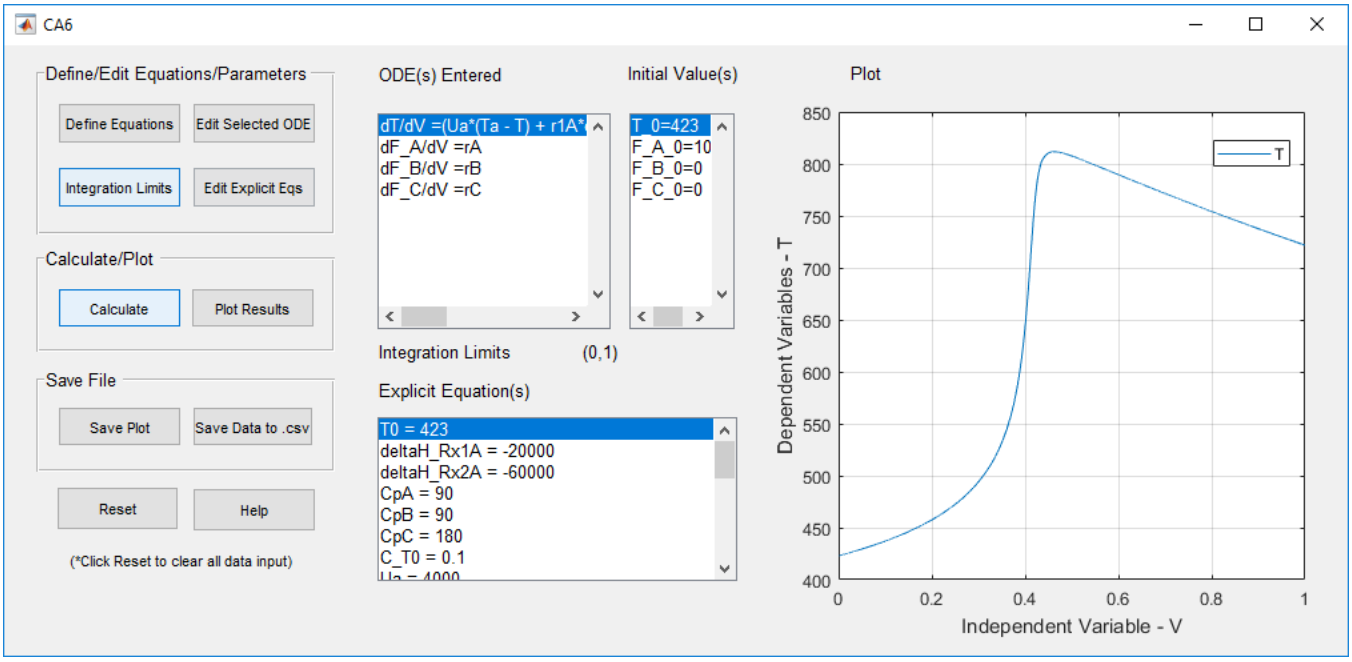


Figure 5: Submission 1 main GUI screenshot with plot of the dependent variable T for the verification case in Appendix A.

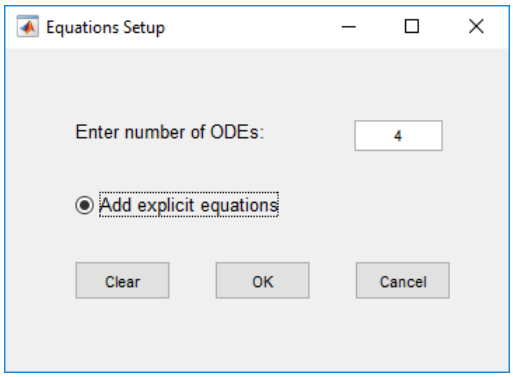


Figure 6: Submission 1 dialog box for setup of the number of equations.

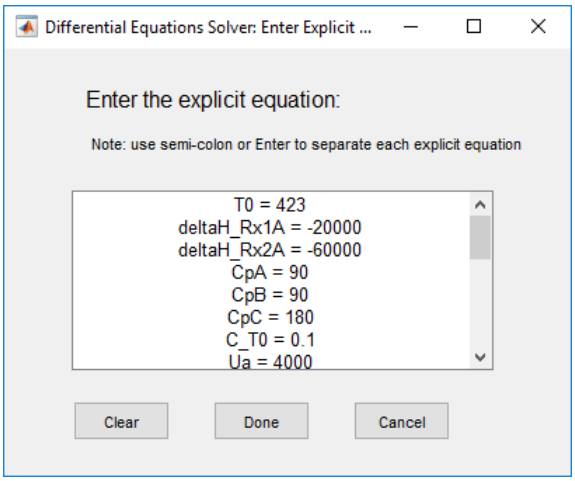


Figure 7: Submission 1 dialog box to enter the explicit equations.

plotted two different sets of the dependent variables in Figs. 5 and 12 to see the curves clearly on their different scales.

3.1.4 Program Shortcomings. One major defect of this program is that after the number of ODEs is defined at the very beginning, it cannot be changed unless the user redefines the whole system of equations. This program did not utilize the MATLAB GUI `handles` structure for passing arguments between functions. The deficiency related to not being able to edit the number of equations could be compensated by adding another function to manipulate the master `ode_eqs.mat` file, which stores the expressions for all the equations. This could be modified carefully by adding another ODE or deleting one or several existing ODEs. Alternatively, the program could be restructured to utilize the `handles`, in which case an update to the number of ODEs would not be as tricky to implement.

Another defect is that the program is not capable of accepting explicit equations in arbitrary order as in POLYMATH, meaning that a parameter in an explicit equation must be defined before it is used. This requires extra work from the users as all the explicit equations have to be listed in a certain order. This is how MATLAB reads codes, so this is not a serious problem. For possible solutions to this problem, see Submission 2 presented in Section 3.2.

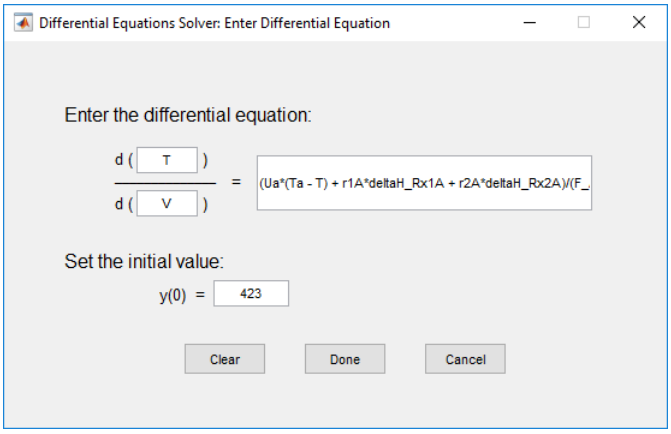


Figure 8: Submission 1 dialog box to enter a differential equation.

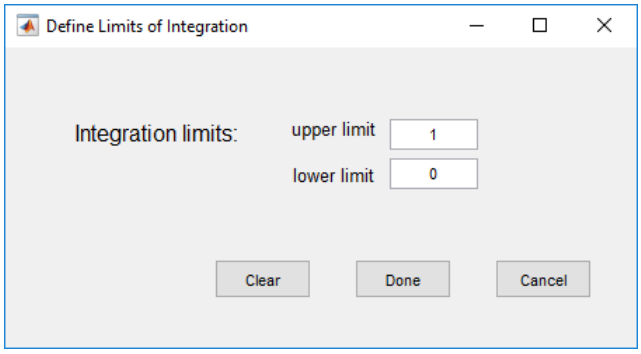


Figure 9: Submission 1 dialog box to define limits of integration.

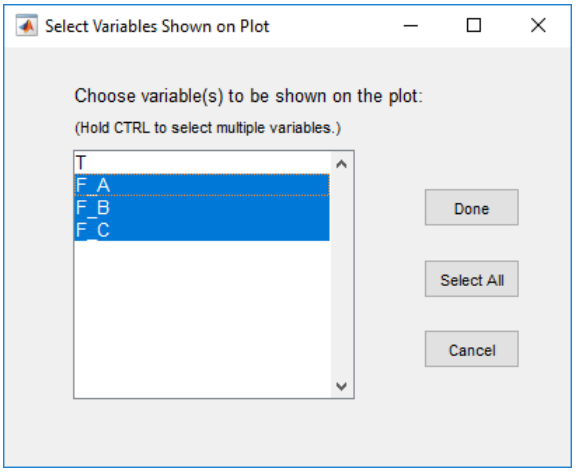


Figure 10: Submission 1 dialog box to select which variables to display on the plot.

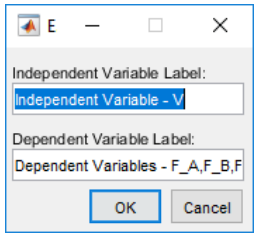


Figure 11: Submission 1 dialog box to edit axes labels starting from default axes labels.

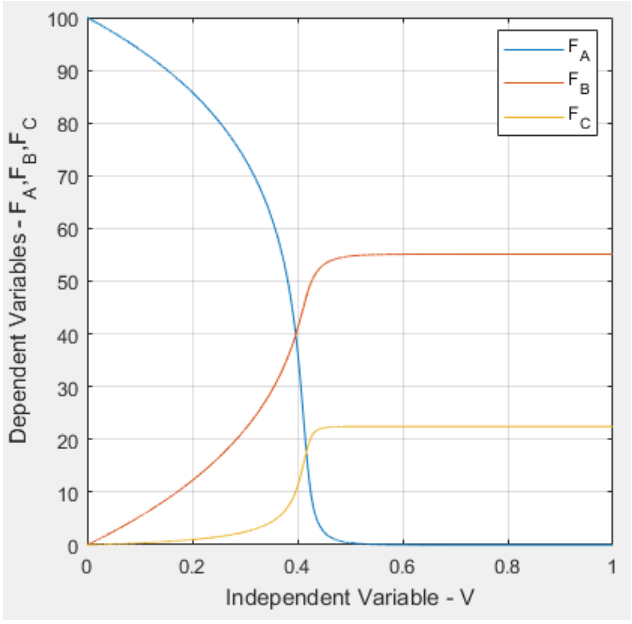


Figure 12: Submission 1 plots of the dependent variables F_A, F_B, and F_C for the verification case in Appendix A.

When the user modifies an ODE, only one ODE from the list box can be selected at one time. Also, items in the `Initial Value(s)` list box can be selected; however, they are not related to any push buttons. In a future version, a new GUI could be added to allow the user to edit initial values based on the selected line in the `Initial Value(s)` list box independent of editing the corresponding ODE.

3.2 Submission 2

3.2.1 Overview. Submission 2 goes beyond the scope of the project requirements by allowing the user the capability to freely edit equations in the GUI and to enter equations in any order (Fig. 13). Allowing the text to be edited enables the user to wholesale copy text in order to share, save, and enter equations. Additionally, if an error is made in entering the equations into the dialog boxes, the user can quickly fix it by editing the text directly. Furthermore, if the user is comfortable, they can type their equations directly without using the dialog boxes. This is very consistent with the capabilities of POLYMATH.

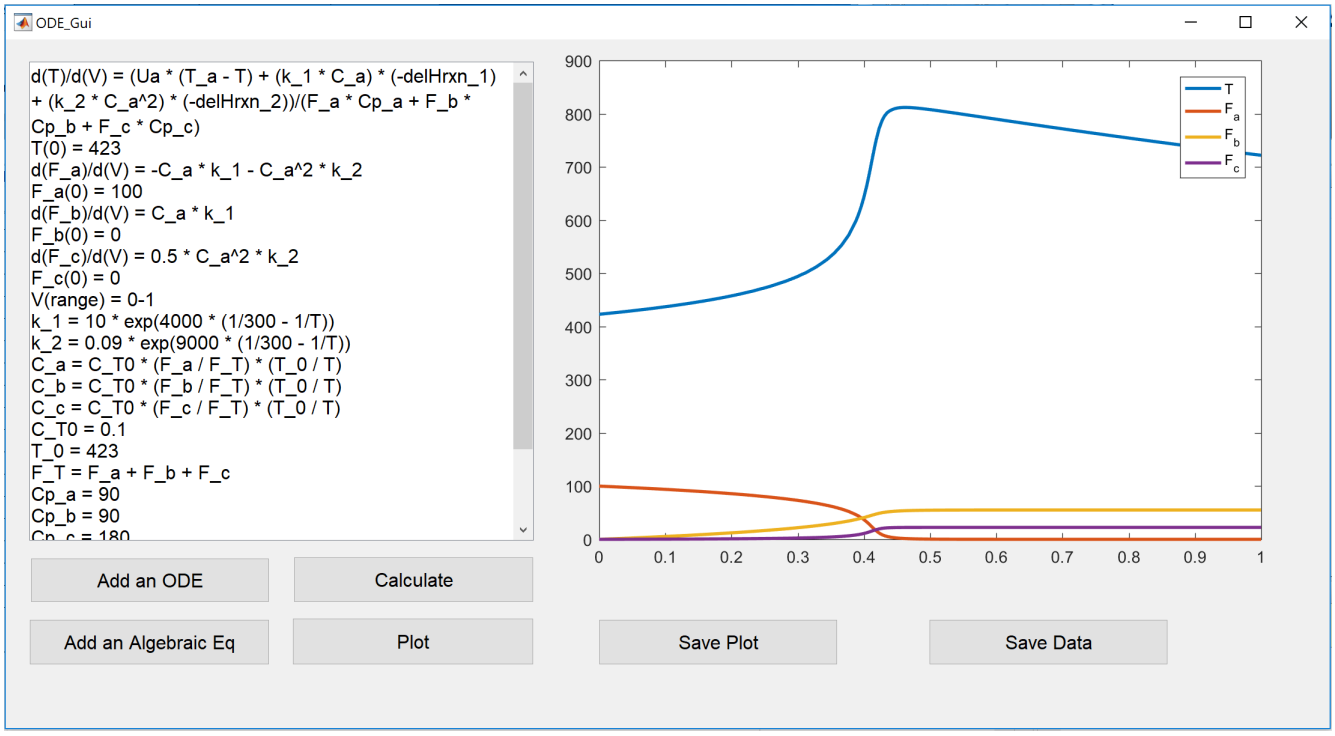


Figure 13: Submission 2 main GUI screenshot.

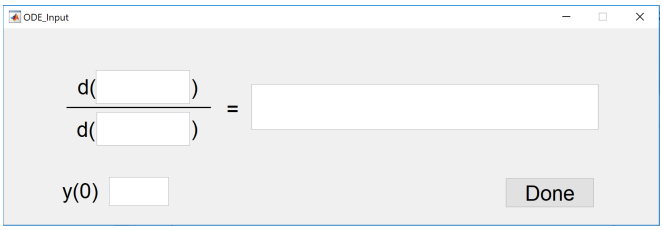


Figure 14: Submission 2 dialog box for entering ODEs.

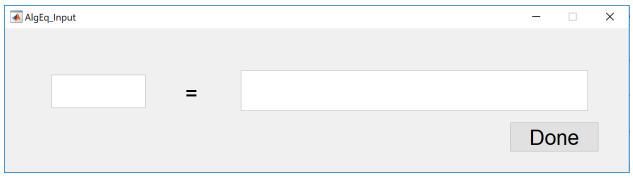


Figure 15: Submission 2 dialog box for entering algebraic equations.

3.2.2 GUI Description. The main GUI is composed of one editable textbox, six push buttons, and one axes (plot) object (Fig. 13). The user can provide inputs through the push buttons in much the same manner as in Section 3.1 through a GUI for defining a differential equation (Fig. 14) and a GUI for defining a single explicit algebraic equation (Fig. 15). The axes object is used to display plots of some or all of the dependent variables as functions of the independent variable (Fig. 13 shows all of the dependent variables selected). The editable textbox is described in detail in the remainder of this section.

3.2.3 Equation Parsing. Properly supporting an editable textbox for equation entry presents some key issues. If the text displayed in the app cannot be edited, then input can be gathered solely from dialog boxes in a very structured manner such as in Section 3.1. Since an editable textbox is a much less structured form of input, interpreting the input becomes a major challenge. The first step in handling the input is to parse the text and convert the text into a structured format. The second step is to reorder the equations so that each equation is only dependent on either no equations or only previously defined equations. This is a requirement because MATLAB requires a variable to be defined before it can be used. The GUI does have dialog boxes that can be used to enter equations (Figs. 14–15); however, these dialog boxes do little more than formatting and inserting the appropriate text into the editable textbox as a template for the user to see how to edit the text.

To parse the text entered into the textbox of the GUI, a custom function `ParseEq.m` accepts a string as an argument and returns a cell array that contains 4 elements. The four elements are the name of the variable the equation solves for, the independent variable associated with the equation if the equation is an ODE, a structure containing the results of parsing the right hand side of the equation, and the type of equation. The right hand side of the equation is returned in two parts: a string of code that can be executed to solve for the dependent variable and a list of variables that need to be defined before the code can be evaluated.

At the heart of parsing the entered text is the use of regular expressions. Regular expressions are useful for finding specific sequences of text. Regular expressions make use of wildcards, whitespace characters, alphanumeric ranges, and more to create a very powerful and flexible syntax for matching text. For example, in the GUI the left hand side of an ODE is in the form of `d(y)/d(t)`, where y and t can be any variables. The regular expression 'd\(\w*\)\/d\(\w*\)' will match the left hand side of any correctly entered ODE.

The first step in parsing the equations is to break the equation into two parts for the left- and right-hand sides. This is done by searching for the '=' character and taking either side as separate strings. Of the two sides, the left-hand side is evaluated first. The left-hand side of each equations has a specific structure–the exact form depends on if the equation is an algebraic equation, ODE, an initial value, or the range of an independent variable. The structured left-hand side of each equation lends itself to being easily parsed through regular expressions.

The main concept used in parsing the right-hand side of an equation is to build a line of code by following order of operations to identify the calculation to be done first, producing the code to evaluate the operation, and then abstracting the evaluated term from the rest of the equation using a token. For example, the string `` `x + y * z' `` becomes `` `x + #1' ``, where `#1` is placeholder for a structure containing the results of parsing `` `y * z' ``. Parsing the right-hand side becomes more complex when handling parentheses. The approach to handling parentheses is to replace the portion of the equation inside of the parenthesis with a token and then recursively calling the`ParseEq.m` function with the replaced text. The result is that for each nested level of parentheses, the function is recursively called until the innermost level of parentheses is reached and evaluated normally.

3.2.4 Equation Reordering. The problem of ordering the equations is reduced to a problem similar to Gaussian elimination. A matrix is constructed where each row represents a dependent variable, and each column represents a unique variable needed to define a variable. For each dependent variable, the element in the column of each variable needed to define the dependent variable is set to 1. For example, the equations

$$x = 2 \tag{1}$$

$$y = 3 * x \tag{2}$$

$$z = 4 * y + x \tag{3}$$

yield the matrix (with columns and rows labeled for clarity)

	x	y
x	0	0
y	1	0
z	1	1

Each dependent variable that is defined by an ODE has an associated initial value previously defined. Therefore, all of the columns associated with variables defined in ODEs are zeroed out right after creating the matrix, representing the fact that the variable is defined. Each dependent variable has exactly one equation in which the variable will show up on the left hand side. When a row corresponding to a variable is zeroed out, the associated equation is put at the bottom of the list of equations, and the column associated with the variable is zeroed out. This process repeats until no changes occur in the matrix after an iteration.

If the matrix is not completely zeroed out, then any rows that are not entirely zeros represent variables that are not properly defined, and an error is returned to the user containing the variables that are improperly defined. If the entire matrix is zeroed out, then the list of equations recorded while zeroing out the matrix is the correct order for the equations so that each equation is only dependent on the previous equation.

4 CONCLUSIONS

The project assignment has been used for 21 total students across two offerings of the Applied Numerical Computing elective course at Oklahoma State University. The project has been challenging and thought provoking for the students in the course without being unreasonable and overly time-consuming. Each student has typically visited the instructor's office hours more than once over the one month time period allotted for the project. The instructor has offered assistance with debugging and brainstorming and implementing approaches. The most challenging aspect of the project for most students is connecting the input from a subsidiary GUI window back to the main GUI window. The submissions described here provide two different methods for doing this, and the instructions and tips from the instructor in the assignment and in Fig. 4 suggest another alternative using `.mat` files and cell arrays. Students are encouraged to discuss ideas with their classmates, but the project must be an individual effort. The vast majority of the students have earned an A on the project (all who started early enough to complete all of the required components, including the verification cases). The students have given the project a positive reception as they can clearly see how it connects the prior course content related to numerical solution of systems of ODEs and development of GUIs for scientific applications. The project detailed in this paper can be easily integrated into a variety of computational science and engineering elective or required courses. The content is approachable for both senior undergraduates and graduate students from a variety of disciplines given sufficient background in MATLAB programming and GUI design.

Additional cases studies could be used to adapt the project to other disciplines such as numerical methods, computational physics or chemistry, mathematical biology, and other fields of engineering. These case studies could readily be developed from textbook examples in these fields or published modeling studies such as [1] in petrochemical manufacturing, [7] in computational pharmacology, and [24] in mathematical biology.

A VERIFICATION CASE

For the verification case study, a system of ODEs is used as defined and solved in an example in a chemical reaction engineering textbook [4]. The equations describe the mass and energy balances for a pair of gas-phase reactions that occur in a plug flow reactor that is operated non-isothermally:

$$\mathrm{A} \xrightarrow{k_1} \mathrm{B} \qquad -r_{1A} = k_{1A} C_A \tag{4}$$

$$2\,\mathrm{A} \xrightarrow{k_2} \mathrm{C} \qquad -r_{2A} = k_{2A} C_A^2 \tag{5}$$

where A, B, and C are chemical species, r_{ij} is the reaction rate of the ith reaction with respect to the jth species, and k_{ij} is the kinetic rate constant for the ith reaction with respect to the jth species. Pure A is fed at a rate of 100 mol/s, a temperature of 423 K, and a concentration of 0.1 mol/L. The molar flow rates of each species, F_A, F_B, and F_C, and the temperature, T, as functions of the reactor volume, V, are the quantities of interest. Mole balances on each species A, B, and C give the ODEs

$$\frac{dF_A}{dV} = r_A \tag{6}$$

$$\frac{dF_B}{dV} = r_B \tag{7}$$

$$\frac{dF_C}{dV} = r_C \tag{8}$$

where r_i is the net reaction rate of species i. The initial conditions are $F_A(0) = 100$ mol/s, $F_B(0) = 0$ mol/s, $F_C(0) = 0$ mol/s, and $T(0) = 423$ K.

The corresponding elementary rate laws that describe reactions 1 and 2 from (4) and (5), respectively, are

$$r_{1A} = -k_{1A}C_A \tag{9}$$

$$r_{2A} = -k_{2A}C_A^2 \tag{10}$$

where C_A is the concentration of species A. The relative rates are

$$r_{1B} = -r_{1A} = k_{1A}C_A \tag{11}$$

$$r_{2C} = -\frac{1}{2}r_{2A} = \frac{k_{2A}}{2}C_A^2 \tag{12}$$

Equations (9)–(12) are combined to yield the net rates,

$$r_A = r_{1A} + r_{2A} = -k_{1A}C_A - k_{2A}C_A^2 \tag{13}$$

$$r_B = r_{1B} = k_{1A}C_A \tag{14}$$

$$r_C = r_{2C} = \frac{k_{2A}}{2}C_A^2 \tag{15}$$

The gas-phase stoichiometry without pressure drop is used to define the concentration of species A as

$$C_A = C_T(0)\frac{F_A}{F_T}\frac{T(0)}{T} \tag{16}$$

where the total flow rate is defined by

$$F_T = F_A + F_B + F_C \tag{17}$$

The rate constants depend on the temperature through the following Arrhenius functions:

$$k_{1A} = 10\exp\left[\frac{E_1}{R}\left(\frac{1}{T(0)} - \frac{1}{T}\right)\right] \text{ s}^{-1} \tag{18}$$

$$k_{2A} = 0.09\exp\left[\frac{E_2}{R}\left(\frac{1}{T(0)} - \frac{1}{T}\right)\right] \frac{\text{L}}{\text{mol}\cdot\text{s}} \tag{19}$$

The energy balance for the reactor is

$$\frac{dT}{dV} = \frac{Ua(T_a - T) + r_{1A}\Delta H_{Rx1A} + r_{2A}\Delta H_{Rx2A}}{F_A C_{P_A} + F_B C_{P_B} + F_C C_{P_C}} \tag{20}$$

The values for the remaining parameters representing physical constants are listed in Table 1.

To summarize, the system of ODEs for the verification case is given by (6)–(8) and (20) for $\frac{dF_A}{dV}$, $\frac{dF_B}{dV}$, $\frac{dF_C}{dV}$, and $\frac{dT}{dV}$, describing the molar flow rates of species, A, B, and C, in mol/s and temperature, T, in K in a non-isothermal plug flow reactor. The reactions are at steady-state but vary spatially along the volume of the reactor, hence V is the independent variable. The explicit equations needed to complete the system of equations are given in (9)–(19) and Table 1.

Table 1: Values of parameters for the verification case study.

Variable	Value	Units
E_1/R	4000	K
E_2/R	9000	K
$C_T(0)$	0.1	mol/L
ΔH_{Rx1A}	-20,000	J/(mol of A reacted in reaction 1)
ΔH_{Rx2A}	-60,000	J/(mol of A reacted in reaction 2)
C_{P_A}	90	J/mol· K
C_{P_B}	90	J/mol· K
C_{P_C}	180	J/mol· K
Ua	4000	J/m^3· s· K
T_a	373	K

REFERENCES

[1] J. Ancheyta-Juarez and J. A. Murillo-Hernandez. 2000. A simple method for estimating gasoline, gas, and coke yields in FCC processes. *Energy and Fuels* 14 (2000), 373–379.

[2] J. Carver and G. K. Thiruvathukal. 2013. Software engineering need not be difficult. *Workshop on Sustainable Software for Science: Practice and Experiences, SuperComputing Conference* (2013). http://dx.doi.org/10.6084/m9.figshare.830442.

[3] B. Ekmekci, C. E. McAnany, and C. Mura. 2016. An introduction to programming for bioscientists: A Python-based primer. *PLOS Computational Biology* 12 (2016), e1004867.

[4] H. S. Fogler. 2011. *Essentials of Chemical Reaction Engineering* (1st. ed.). Prentice Hall, Boston, MA.

[5] H. S. Fogler and M. Tikmani. [n. d.]. Polymath tutorial on Ordinary Differential Equation Solver. ([n. d.]). Retrieved January 4, 2018 from http://umich.edu/~elements/5e/tutorials/ODE_Equation_Tutorial.pdf

[6] A. N. Ford Versypt. 2017. Choose Your Own Kinetics Adventure: Student-Designed Case Studies for Chemical Reaction Engineering Course Projects. *Transactions and Techniques in STEM Education* In Press (2017).

[7] A. N. Ford Versypt, G. K. Harrell, and A. N. McPeak. 2017. A pharmacokinetic/pharmacodynamic model of ACE inhibition of the renin-angiotensin system for normal and impaired renal function. *Computers and Chemical Engineering* 104 (2017), 311–322.

[8] T. Hall and J-P Stacey. 2009. Chapter 2: Designing Software. In *Python 3 for Absolute Beginners*. Springer, New York.

[9] D. M. Hamby. 1994. Review of techniques for parameter sensitivity analysis of environmental models. *Environmental Monitoring and Assessment* 32 (1994), 135–154.

[10] A. K. Hartmann. 2015. *Big Practical Guide to Computer Simulations* (2nd ed.). World Scientific, Hackensack, NJ.

[11] F. M. Hemez and J. R. Kamm. 2008. A brief overview of the state-of-the-practice and current challenges in solution verification. In *Computational Methods in Transport: Verification and Validation*, F. Graziani (Ed.). Springer-Verlag, Berlin, 229–250.

[12] D. J. Higham and N. J. Higham. 2005. *MATLAB Guide* (2nd ed.). Society for Industrial and Applied Mathematics, Philadelphia.

[13] J. M. Kinder and P. Nelson. 2015. *A Student's Guide to Python for Physical Modeling*. Princeton University Press, Princeton, NJ.

[14] J. Kiusalaas. 2005. *Numerical Methods in Engineering with Python*. Cambridge University Press, New York.

[15] C. S. Lent. 2013. *Learning to Program with MATLAB: Building GUI Tools*. Wiley, Hoboken, NJ.

[16] MathWorks. [n. d.]. App Building. ([n. d.]). Retrieved February 15, 2018 from https://www.mathworks.com/help/matlab/gui-development.html

[17] MathWorks. [n. d.]. Create a Simple App Using GUIDE. ([n. d.]). Retrieved February 15, 2018 from https://www.mathworks.com/help/matlab/creating_guis/about-the-simple-guide-gui-example.html

[18] C. B. Moler. 2004. *Numerical Computing with MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia.

[19] Oliver Woodford. 2013 (accessed November 12, 2016). *export_fig: A MATLAB toolbox for exporting publication quality figures.* https://github.com/altmany/export_fig.
[20] J. M. Osborne, M. O. Bernabeu, M. Bruna, B. Calderhead, J. Cooper, N. Dalchau, S.-J. Dunn, A. G. Fletcher, R. Freeman, D. Groen, B. Knapp, G. J. McInerny, G. R. Mirams, J. Pitt-Francis, B. Sengupta, D. W. Wright, C. A. Yates, D. J. Gavaghan, S. Emmott, and C. Deane. 2014. Ten Simple Rules for Effective Computational Research. *PLOS Computational Biology* 10 (2014), e1003506.
[21] A. Prlic and J. B. Procter. 2012. Ten simple rules for the open development of scientific software. *PLOS Computational Biology* 8 (2012), e1002802.
[22] P. J. Roache. 1998. Verification of codes and calculations. *AIAA Journal* 36, 5 (1998), 696–702.
[23] K. Rother, W. Potrzebowski, T. Puton, M. Rother, E. Wywial, and J. M. Bujnicki. 2011. A toolbox for developing bioinformatics software. *Briefings in Bioinformatics* 13, 2 (2011), 244–257.
[24] S. M. Ruggiero, M. R. Pilvankar, and A. N. Ford Versypt. 2017. Computational modeling of tuberculosis granuloma activation. *Processes* 5 (2017), 79.
[25] M. Shacham, M. B. Cutlip, and M. Elly. [n. d.]. POLYMATH Software. ([n. d.]). Retrieved January 4, 2018 from http://www.polymath-software.com/
[26] V. Stodden and S. Miguez. 2013. Best practices for computational science: software infrastructure and environments for reproducible and extensible research. (2013). http://dx.doi.org/10.2139/ssrn.2322276.
[27] G. Wilson. 2016. Software Carpentry: lessons learned [version 2]. *F1000Research* 3 (2016), 62. https://doi.org/10.12688/f1000research.3-62.v2
[28] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson. 2014. Best Practices for Scientific Computing. *PLOS Biology* 12 (2014), e1001745.

Modelling the Effects of Star Formation with a Volumetric Feedback Model

Claire Kopenhafer
Michigan State University
East Lansing, Michigan
kopenhaf@msu.edu

Brian W. O'Shea
Michigan State University
East Lansing, Michigan
oshea@msu.edu

ABSTRACT

We implemented two new models (an original and a revised) for star formation and supernova feedback into the astrophysical hydrodynamics code Enzo. These models are designed to efficiently capture the bulk properties of galaxies and the influence of the circumgalactic medium (CGM). Unlike Enzo's existing models, these do not track stellar populations over time with computationally expensive particle objects. Instead, supernova explosions immediately follow stellar birth and their feedback is deposited in a within a predefined volume of cells. Our models were tested using simulations of Milky Way-like isolated galaxies, and we found that neither model was able to produce a realistic, metal-enriched CGM. Our work suggests that volumetric feedback models are not sufficient replacements for particle-based star formation and feedback models.

1 INTRODUCTION

The large-scale structure of the universe is dominated by dark matter and its resulting gravitational potential, and individual galaxies are no exception. A galaxy's baryonic components, its stars, gas, and dust, sit within the potential well of its accompanying dark matter halo. While the baryons emit all the light we observe from galaxies, galaxy masses are dominated by the dark matter. Stars form near the center of the halo, where the gravitational potential is the strongest. In a spiral galaxy like our Milky Way, which has a mass of roughly 10^{12} $M_\odot$, most stars and their interstellar medium form a disk structure because of their angular momentum. Away from the center of the dark matter halo, the baryon density is too low to form stars. Instead, there exists the diffuse, multiphase gas of the circumgalactic medium (CGM). Despite its low density, the CGM is estimated to contain roughly half of the galaxy's total baryons [Tumlinson et al. 2017; Werk et al. 2014]. It is also believed to substantially impact the bulk properties of the galaxy, such as its star formation history [Voit et al. 2015a].

While galaxies evolve over time, at a given age of the universe they are observed to follow strong scaling relations. Properties such as luminosity and metallicity[1] are highly correlated to the mass of galaxy's dark matter halo [Graves et al. 2009; McConnachie 2012; McGaugh 2005]. These correlations imply galaxies self-regulate themselves to a state that depends primarily on the mass of the halo and its associated system.

The CGM is believed to be a key player in this process. One possible mechanism is that of "precipitation" [Voit et al. 2015a] in which CGM gas cools and falls deeper in to the potential well and onto the disk. This cold, dense infall encourages star formation. When massive stars (with mass greater than about 8 $M_\odot$[2]) die, they explode as supernovae, ejecting energy and metal-enriched material into their surroundings. This ejecta is known as feedback, as it strongly effects the galaxy as a whole. According to the theory of precipitation, feedback from stellar populations lowers the density of the CGM by pushing gas to larger radii. This in turn increases the timescale on which the gas will cool. Once the cooling timescale exceeds the gas' freefall timescale by a factor of ~ 10 [Voit and Donahue 2015; Voit et al. 2015b,c], cold gas is no longer able to precipitate and cause further star formation. This decreased ability to form stars is known as "quenching."

We would like to examine the plausibility of this theory using numerical simulations; however, current simulations of single galaxies cannot resolve the scales on which individual star formation occurs. Star formation cannot be directly modeled in a galactic context, as we lack the computational resources needed to efficiently resolve the wide range of spatial and temporal scales that separate star formation and overall galactic dynamics. Instead, the effects of star formation and feedback can be modeled with heuristic "subgrid" models. If galaxy self-regulation by precipitation is to be considered a plausible explanation for galaxy behavior, galaxy models should be robust to the exact model of star formation and feedback chosen. These models, which involve tracking particles throughout the simulation, also add a good deal of computational expense. Some effects of precipitation, such as the reproduction of the mass-metallicity relation predicted in [Voit et al. 2015a], only affect the bulk properties of the galaxy. In this case it would be preferable to have a simpler, more idealized model of star formation and feedback.

The development of such a model, with the goal of testing the scaling relations predicted in [Voit et al. 2015a], is the focus of this work. Our models were implemented within Enzo. This code, as well as its existing treatment of star formation and feedback using particles, is described in Section 2.1. As a computationally cheaper alternative to particles, we employ a volumetric approach to feedback, which is described in Section 2.2. Massive stars are assumed to immediately result in supernovae in order to avoid tracking their ages and masses, as the details of the stellar populations are not the focus of our queries. Two models were developed: an original model, and a revised model. These models are referred to as "volumetric" as

[1] Astronomers refer to elements heavier than helium as "metals". "Metallicity" refers to the metal fraction of a gas, measured relative to the metal content of the sun.

[2] A mass measured in solar masses $M_\odot$ is measured with respect to the mass of our sun.

DOI: https://doi.org/10.22369/issn.2153-4136/9/1/3

they deposit stellar feedback within a predefined simulation volume. The revised model is an attempt to alter some undesired behaviors that were observed in the original model. Section 2.3 covers the details of our simulation initial conditions and parameter sets. Next, Section 3 covers the simulation behavior over time (3.1 and 3.2) including the effects of parameter variations. Ultimately, neither model behaved as desired; exactly how the models failed to meet the mark and possible reasons why are discussed in Section 4. As this work resulted from a student project, Section 4.1 is where the student's experiences and challenges are discussed. A summary and concluding remarks are offered in Section 5.

2 METHODS

Before discussing the specific model employed in this work (Section 2.2), we first introduce the code base that is used for our simulations in Section 2.1. Then, in Section 2.3, the initial conditions and parameter variations are described.

Table 1 contains a list of all relevant simulation parameters. Those with a symbol listed are used elsewhere in this paper; those without are included for reference. A list of values indicates a parameter was varied between different simulation runs; see Table 2 for the exact combinations and more details about the runs.

Parameter Name	Symbol	Value
Halo Mass		9×10^{11} $M_\odot$
NFW Concentration		12
Disk Gas Mass	M_G	5×10^{10} $M_\odot$
Disk Gas Temperature		2×10^5 K
Galaxy Radial Scale Height	r_s	3.0 kpc
Galaxy Vertical Scale Height	z_s	0.4 kpc
Galaxy Extent (# of Scale Heights)	κ	4
Feedback Region Extent	κ_{FB}	[2, 3]
Star Formation Density Threshold	ρ_{SF}	1 cm^{-3}
Feedback Density Threshold	ρ_{FB}	0.01 cm^{-3}
Star Formation Temperature Thres.	T_{SF}	$10^{4.5}$ K
Feedback Temperature Threshold	T_{FB}	10^7 K
Star Formation Efficiency	ϵ_{SF}	0.01
Star Formation Timescale	τ_{SF}	10 Myr
Stellar Mass Removal Factor	η	[1, 100]
Energy SNe Feedback Efficiency	ϵ_{FB}	10^{-5}
Mass SNe Feedback Efficiency	$\epsilon_{FB,M}$	0.25
Metal SNe Feedback Efficiency	$\epsilon_{FB,Z}$	0.02
Kinetic Fraction	f_k	[0, 0.25, 0.5, 0.75, 1]
Returned Mass Fraction	η_M	0.25
Returned Metal Fraction	η_Z	0.2

Table 1: Relevant simulation parameters. Parameters accompanied by a symbol are used elsewhere in this work. Values in brackets correspond to simulation variations; see Table 2.

2.1 Enzo

The models discussed in Section 2.2 below were implemented in Enzo [Bryan et al. 2014][3], a multi-physics hydrodynamics code designed to simulate astrophysical problems from cosmology to plasma turbulence. Enzo stores data as either particles or Cartesian grid cells. For grid data, Enzo employs adaptive mesh refinement (AMR) to balance accuracy and computational efficiency by improving grid resolution in user-defined areas of interest. An Enzo user may define these areas using a variety of criteria, such as baryon or dark matter density, the presence of shocks, or a geometric region. At the core of Enzo's physics are gravity and Eulerian (magneto)hydrodynamics. Gravity acts on both particle and grid cell data. For ordinary hydrodynamics, which is of interest to this work, there are two solvers: the ZEUS finite difference method (adapted from [Stone and Norman 1992]) and the finite-volume piecewise-parabolic method (PPM) [Colella and Woodward 1984].

Other important physics in Enzo includes radiation transport, star formation and feedback, primordial chemistry, and radiative cooling. Support for the latter two is provided by the GRACKLE library [Smith et al. 2017][4]. Natively, star formation and feedback are modeled using particle objects instead of grid cells. These particles are not constrained to the grid where gas information is stored. Instead, they are free to move about the simulation volume. A single particle represents an entire stellar cluster. These particles track the mass distribution of stars in the cluster, as well as their lifetimes. All stars in the modeled cluster are assumed to form at the same time, but more massive stars live shorter lives. It is known from observations that stars with main-sequence mass $\gtrsim 8$ $M_\odot$ die in supernova explosions, ejecting energy and metal-enriched material into the surrounding environment. This is referred to as feedback. In Enzo, feedback from a particle is deposited into the gas in nearby grid cells. Several different star formation and feedback models are available, which define when particles are created and the mechanism by which feedback is deposited into the surrounding grid cells.

2.2 Volumetric Feedback Models

In our model, star formation and feedback can only occur within a cylinder of user-defined height and radius. This cylindrical region is located at the center of the domain, and encompasses the central region of the initial galactic disk. Within this domain, grid cells are flagged as being either star-forming or feedback-only depending on their density and temperature. If a cell has density $\rho > \rho_{SF}$ and temperature $T < T_{SF}$, where ρ_{SF} and T_{SF} are parameters, it is flagged as a star-forming cell. If a cell has $\rho_{FB} < \rho < \rho_{SF}$ and $T < T_{FB}$, where ρ_{FB} and T_{FB} are parameters, it is flagged as feedback-only. These two categories are constructed to be mutually exclusive, and most cells are not flagged at all. The temperature parameters should be set such that $T_{FB} > TSF$. The parameter T_{FB} also controls the energy budget for feedback (see Section 2.2.2).

Figure 1 shows what this flagging looks like for the density and temperature thresholds in Table 1. The grayscale shows a face-on density slice through the midplane of the disk. Overlaid in cyan and yellow are the star-forming and feedback-only cells, respectively. The red circle indicates the boundary of the star formation and feedback region. The scale bar corresponds to the radius of this region.

[3] http://enzo-project.org/
[4] https://grackle.readthedocs.org/

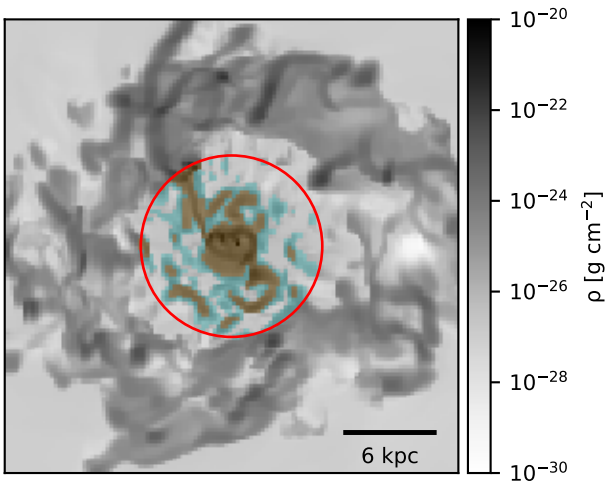


Figure 1: Density slice showing an example of star-forming (cyan) and feedback-only (yellow) cells within the feedback region (red circle; scale bar). The slice shows the galactic disk face-on, and passes through the midplane.

In the subsections below, we describe how star formation (2.2.1) and feedback (2.2.2) are handled with reference to these cell types. For each process, we first describe the treatment of the original model. Problems with the results of the original model (see the end of Section 3.1.1 and Section 4) induced some revisions which will be discussed later in each section.

2.2.1 Star Formation and Stellar Death. Once star-forming cells have been flagged, their total mass M_{SF} is used to calculate a mass of stars ΔM_* formed during that time step Δt using the following formula:

$$\dot{M}_* = \epsilon_{SF} M_{SF}/\tau_{SF}, \quad (1)$$

$$\Delta M_* = \eta \dot{M}_* \Delta t. \quad (2)$$

The star formation efficiency is ϵ_{SF} and the formation timescale is τ_{SF} (see Table 1). Stars are never directly modeled by either the original or revised schemes, because the goal of these models was to avoid the computational expense of explicitly including stellar populations via particles. Instead, only the effects of star formation on the surrounding gas are modeled: ΔM_* worth of gas is removed from all star-forming cells. The amount removed from an individual cells is proportional to its fraction of the total star-forming cell mass M_{SF}.

The parameter η in Equation 2 allows mass to be removed from the simulation in excess of what is consumed by star formation. This mimics the ionization of surrounding gas that occurs when stars form, which prevents further star formation in the immediate area of the new stars. This removed mass is negligible compared to the mass of gas in the disk. Likewise, it has a negligible effect on the gravitaional potential, which is dominated by the dark matter and remaining gas. For the original model, $\eta = 1$. In the revised model, η is a tunable parameter (Table 1).

When stellar populations in Enzo are modeled with particles, these particles keep track of the age and individual masses of their constituents. For simplicity in these models, stellar death is treated as though it immediately follows stellar birth. Not all stars are massive enough to result in supernova explosions; the efficiency parameters ϵ_{FB}, $\epsilon_{FB,Z}$ and $\epsilon_{FB,M}$ are chosen to reflect the fraction that do (see Table 1). The energy, mass, and metal returned by these supernovae is calculated in bulk:

$$\begin{aligned} \Delta E &= \epsilon_{FB} \Delta M_* c^2, \\ \Delta M &= \epsilon_{FB,Z} \Delta M_*, \\ \Delta Z &= \epsilon_{FB,M} \Delta M_*. \end{aligned} \quad (3)$$

For each time step, ΔE, ΔM, and ΔZ are added to "reservoirs" from which feedback energy and material are drawn. These reservoirs are maintained throughout the simulation, allowing past star formation to have some affect on current feedback.

2.2.2 Supernova Feedback. In these models, feedback constitutes the addition of energy, mass, and metal to a certain selection of cells, which we will refer to as "feedback cells." For the original model, feedback cells refer to the feedback-only cells discussed above in Section 2. In the revised model, feedback cells refer to *both* the star-forming and feedback-only cells. Unflagged cells do not receive feedback.

An "energy budget" e_b is calculated for each feedback cell. This is the amount of energy that would be needed for $(1 - f_k)e_b$ to raise the temperature of the cell to T_{FB}, where f_k is the kinetic fraction. The interplay between this energy budget and the feedback algorithm will be discussed later. Recall from Section 2.2 that all flagged cells have $T < T_{FB}$. Feedback will only proceed if there are feedback cells to receive it, and if the sum of e_b for all the feedback cells is less than the amount of energy E_{res} in the energy reservoir.

For each cell, the energy budget is divided into thermal and kinetic energy based on the cell's height $|z|$ from the galactic midplane. In the original model, every cell receives thermal energy e_t of exactly $(1 - f_k)e_b$ (marked by a dashed line). Only the outermost layer of cells receives kinetic energy $e_k = f_k e_b$. For the revised model, the kinetic energy is given by $e_k = f_k \tanh(2|z|/z_s)$ and the thermal energy is $e_t = e_b - e_k$. The thermal energy is never less than $(1 - f_k)e_b$ (shown by the dashed line).

Mass and metal are also returned during feedback, and are proportional to the total amount of energy subtracted from the energy reservoir:

$$\Delta M_{ret} = \eta_M e_b / E_{res}, \quad (4)$$

$$\Delta Z_{ret} = \eta_Z e_b / E_{res}. \quad (5)$$

This ensures that all three reservoirs - energy, mass, and metal - remain proportional to each other.

2.3 Simulations

Our simulations are of a single, Milky Way-like disk galaxy. The galaxy is isolated in a (1 Mpc)^3 box whose boundaries are periodic for baryons but not for gravity. The galactic disk is constructed as a cylinder of radius κr_s and height κz_s (see Table 1 for symbol definitions; the values chosen for r_s and z_s are consistent with Kim et al. [2016]) placed in a Navarro-Frenk-White, or NFW, dark matter halo (Navarro et al. [1996]; see Table 1). The disk's initial temperature is

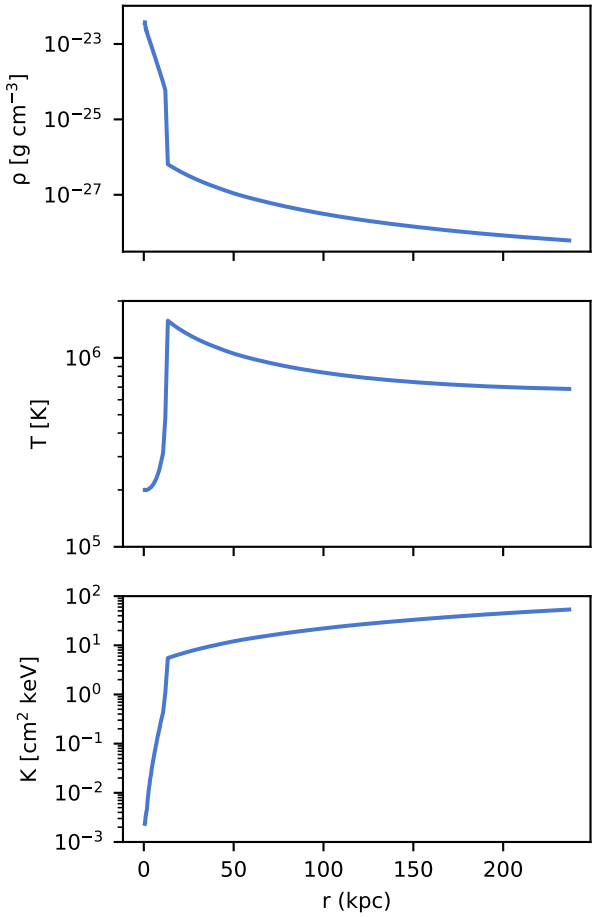


Figure 2: Radial profiles of the simulation initial conditions. Top is density, middle is temperature, and bottom is entropy (the adiabatic invariant; see text). Profiles are measured from the center of the galaxy and extend out to 250 kpc (kiloparsecs). They show the average value at that radius.

uniform, while its density follows the double-exponential profile used in the AGORA simulation suite [Kim et al. 2016]:

$$\rho(x, y, z) = \frac{M_G}{4\pi r_s^2 z_s} e^{-r/r_s} e^{-|z|/z_s}. \quad (6)$$

Surrounding the disk is gas that follows the cored entropy profile $K(r) = 4 + 20(r/100 \text{ kpc})^{1.15}$ [Voit et al. 2017]. This profile is used in conjunction with the assumption of hydrostatic equilibrium to calculate the density and temperature profiles for the CGM. As used here, "entropy" refers to the adiabatic invariant $K = T/\rho^{\gamma-1} \approx T/n_e^{2/3}$, where n_e is the electron number density and the ratio of specific heats is $\gamma = 5/3$. See Figure 2 for radial profiles of the initial density, temperature, and entropy out to a radius of 250 kpc. In addition to the feedback models discussed above, we use Enzo's gravity and PPM hydrodynamic solver, as well as radiative cooling with the GRACKLE library [Smith et al. 2017].

Model	f_k	κ_{FB}	η
Original	0.5	2	1
	0	2	1
	0.25	2	1
	0.75	2	1
	1	2	1
	0.5	3	1
Revised	0.5	2	100

Table 2: Parameter variations; each line corresponds to a different simulation. The first set listed for each model is fiducial parameter set for that model.

In total, seven simulations were run: six using the original model and one using the revised model. A summary of the parameter variations can be found in Table 2. The parameter set listed next to the model name is the fiducial set for that model. The original fiducial simulation ran for 8 Gyr of simulation time. The other non-fiducial simulations were set to run for 5 Gyr; however, only the model $f_k = 0.25$ reached this mark in a reasonable amount of time. All others were halted because the evolution slowed to the point where progress was minimal, with the simulation timesteps of less than one year. The timestep is constrained by the Courant-Freidrichs-Levy (CFL) condition for stability and accuracy (see Bryan et al. [2014], Section 9). Section 3.1.2 has more detail on where each run was stopped. The fiducial run of the revised model was stopped at 2 Gyr as it was not producing a more realistic galaxy than the original model (see sections 3.2 and 4).

3 RESULTS

The results of the fiducial runs for both models are discussed below. First, we detail the behavior of the original model for its fiducial run, and then discuss parameter variations. Then we discuss the behavior of the revised model's fiducial run. The parameters were not varied for the revised model, because reasonable parameter values did not affect the overall outcome of the simulations.

3.1 Original Model

3.1.1 Fiducial Run. The time evolution of the star formation rate (SFR) for the original fiducial simulation is shown in Figure 3. Note that this plot has been smoothed (original in grey dots) for easier viewing using a Savitsky-Golay scheme with 4th order polynomials and a window of 8001 elements. Times of interest are labeled A–C. These times correspond to the images in Figures 4–6, which show projections of the edge-on disk and its inner CGM in four quantities (clockwise from upper left): density, temperature, metallicity, and radial velocity, with the latter three weighted by density. The projections are through a slab 40 kpc thick and 100 kpc on a side.

There is an initial burst of star formation seen in Figure 3 that is triggered by the initial conditions. After this burst, the rate of star formation drops before rising again. Point A (Figure 4) falls in this lull. Point B (Figure 5) corresponds to the peak of the SFR curve, just before the rate begins to slowly fall. The steady-state behavior of the system is sampled at Point C (Figure 6).

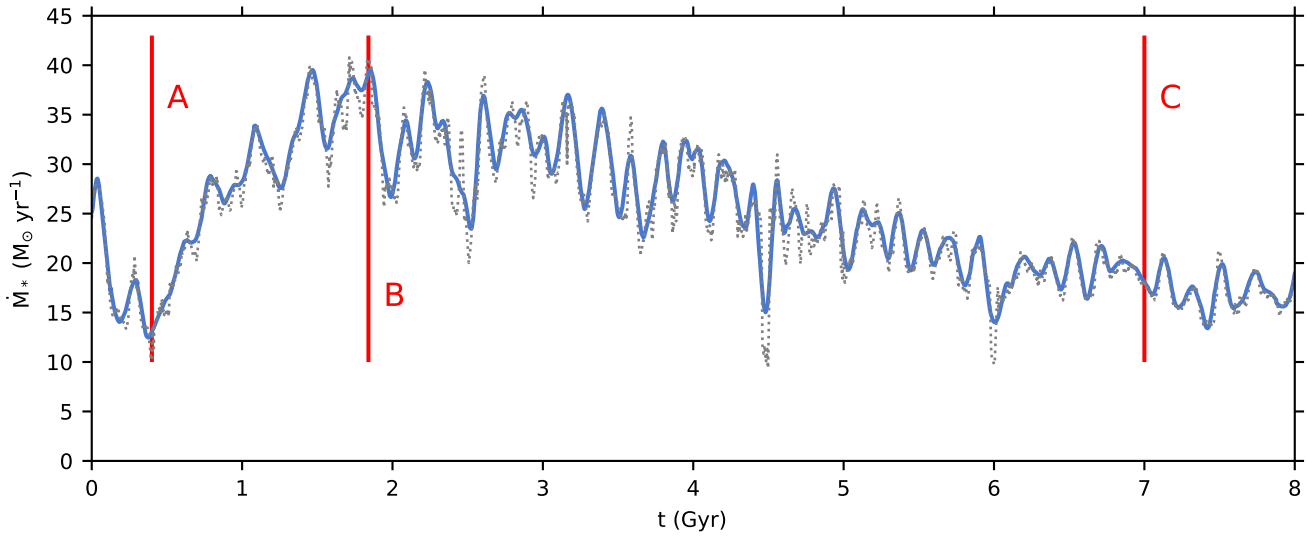


Figure 3: Star formation rate (SFR) over time for the original fiducial run. For easier viewing, the blue line has been smoothed with a Savitsky-Golay scheme; see text The dashed grey line underneath shows the original data. The vertical red lines labeled A–C are at 0.40, 1.84, and 7.00 Gyr, and correspond to Figures 4–6.

Figure 4 shows that at Point A (0.40 Gyr), the cold, dense disk is metal poor, and surrounded by a hot bubble that arose from earlier feedback. There is a large amount of material above and below the disk. Some of this is outflowing gas (coded red in the lower right of Figure 4). The larger outflows are disconnected from the disk, as this material was ejected during earlier times when the SFR and corresponding instantaneous feedback were higher. Newer, smaller outflows are seen closer to the disk; there is less gas being ejected because the SFR is lower. When one simultaneously considers the projections of temperature, metallicity, and radial velocity, infalls of cold, metal-rich gas can be seen at the edges of the largest outflows. This gas comes from even earlier stellar feedback injection. Both infall and outflow are of roughly the same projected metallicity and temperature.

At Point B (1.84 Gyr), we are at end of the peak in the SFR curve from Figure 3. We see from Figure 5 that the CGM has become hotter, but it's metallicity has not increased in the substantial time since Point A. The metallicity of the disk, however, is higher than at A. Highly collimated infalling gas can be seen directly above and below the gas, with bursts of outflowing material to the sides; however, the outflows have a smaller radial extent than in Figure 4.

By the time the simulation reaches Point C, it is is approximately in its steady-state with an SFR of ~ 18 $M_\odot$ yr^1. This is a significantly higher SFR than we expect for a Milky Way-like galaxy. In Figure 6 we see that the disk is has been greatly enriched by metals, and while the CGM has been heated relatively uniformly at this point, it is unenriched. There are no more large-scale outflows, but there is still collimated infall. From Figure 3 we see that the SFR has been decreasing up to this Point; therefore, both the amount of feedback injected and, by extension, the amount of ejected material has also been decreasing. There is less older ejecta to fall back on to the disk, and less star formation and feedback to drive newer outflows. Overall, activity in and around the galaxy is calming down.

Considering Figures 4–6 together, along with insights from examining the evolution over each of the 1 Myr time outputs, several trends emerge. The temperature of the CGM increases over time, as does the metallicity of the disk. The CGM also becomes hotter; however, its metallicity never changes. What is especially noticeable from examining the evolution of the simulation is that all of the material ejected from the disk eventually falls back onto the disk without becoming buoyant in the CGM. Therefore, the CGM remains metal poor, while the disk becomes highly enriched. The outflows are of a lower entropy than the surrounding CGM gas (this is visible in Figures 4–6 via the temperature). This entropy deficiency would explain the lack of buoyancy in the outflowing gas.

3.1.2 Parameter Variations. Five variations on the fiducial parameter set were run for the original model. In four of these runs, the fraction of total energy budget that went into kinetic energy (f_k) was varied. In the fifth, the kinetic fraction remained at the fiducial value of $f_k = 0.5$, while the size of the feedback region κ_{FB} was increased from 2 to 3 scale heights in both z and r.

Figure 7 shows how the star formation rate for all the parameter variations compares to the fiducial value (shown in thin grey dots; the same curve as Figure 3). These SFR curves have been smoothed in the same way as Figure 3, and have been limited to 5 Gyr for clarity. Values near $f_k = 0.5$ (solid lines) did not produce much variation in the SFR curve. Extreme values (dash-dot lines), however, differ greatly from the fiducial run. The fully thermal feedback of the $f_k = 0$ run constantly heated the gas in the disk, making it increasingly less able to form stars. Interestingly, $f_k = 1$ maintains a steadier, if lower, SFR than any of the $0 < f_k < 1$. Lastly, increasing

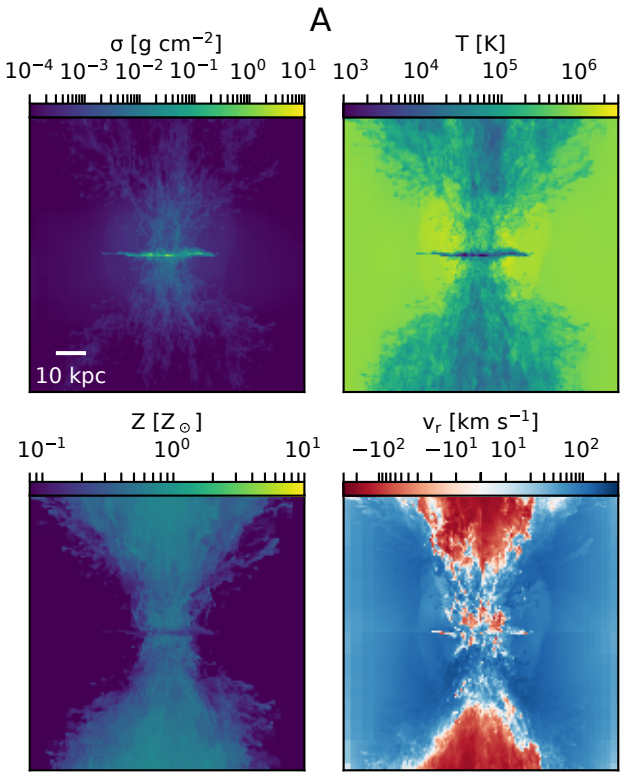


Figure 4: Projections of the original fiducial run at 0.40 Gyr, centered on the edge-on disk and inner CGM. This figure corresponds to Point A in Figure 3. Clockwise from upper left: density; density-weighted temperature, density-weighted radial velocity, and density-weighted metallicity. Negative v_r is outward. The galaxy is actively producing metal-enriched outflows (red), and shows signs of earlier feedback (hot bubble, blue inflows). Each projection is 100 kpc wide and through a slab 40 kpc thick.

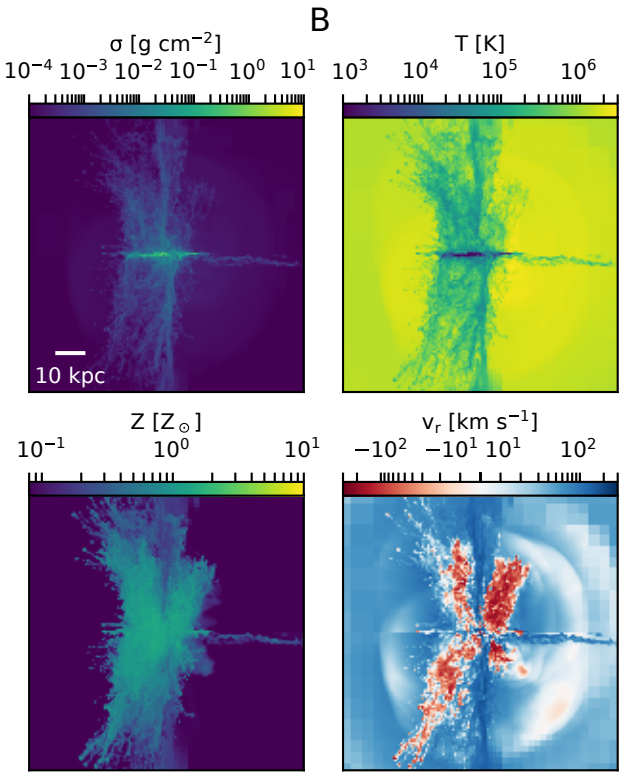


Figure 5: Shows the same quantities as Figure 4, but for Point B of Figure 3 at 1.84 Gyr. The galaxy is at its peak star formation rate, yet there is more infalling gas (dark blue) from previous feedback than recent outflows (red). Both outflows and inflows are the only enriched gas.

the extent of the feedback region, κ_{FB}, leads to an SFR curve with a similar shape to the fiducial run, but with a larger initial burst and stretched in time. This is a reasonable result, as increasing κ_{FB} increases the amount of gas that can form stars.

The $f_k = 0.25$ simulation was able to run for the initially planned length of 5 Gyr, as was $f_k = 1$ (neither was extended to 8 Gyr like the fiducial run). The run with $f_k = 0.75$ was stopped after 1.8 Gyr because of its slow progress; because of the resulting high velocities, the simulation took very small time steps. Similarly, the $f_k = 0$ simulation only ran for about 1 Gyr, as did $\kappa_{FB} = 3$. For the former, the elevated temperatures resulting from the completely thermal feedback greatly restricted the size of the time step that could be taken.

From Figure 7, it appears that any $0 < f_k < 1$ will produce an SFR curve like Figure 3. The exact value has more of an impact on the evolution of the simulation than its behavior. Varying the size of the feedback region stretches the SFR curve but slows the simulation's evolution. Additionally, runs with $f_k = 0.25$, $f_k = 0.75$, and $\kappa_{FB} = 3$ reproduce the same overall behavior of the fiducial run: increasing metallicity in the disk but not the CGM, increasing temperature and density of the CGM (except the density for $f_k = 0.25$), and all outflowing gas eventually falling back on to the disk.

3.2 Revised Model

We will now consider the results of the revised feedback model. Like Figure 3, Figure 8 shows the star formation rate over time for the revised fiducial model. This model shows much less star formation overall, as expected: with this revised model, cold gas is being removed in excess of that used to form stars, diminishing the galaxy's gas reserves. This removal of extra gas is analogous to how cold, dense gas in stellar environments is destroyed by processes such as OB associations and Type II supernovae.

As before, times of interest are marked by red lines marked D and E, at 0.25 and 1.15 Gyr, respectively. Figure 9 corresponds to Point D, showing the simulation during the peak of a small star burst. Point E marks a quiescent period for the galaxy, and corresponds to Figure 10.

We see in Figures 9 and 10 that the metallicity of the disk increases over time, while that of the CGM remains constant. This is

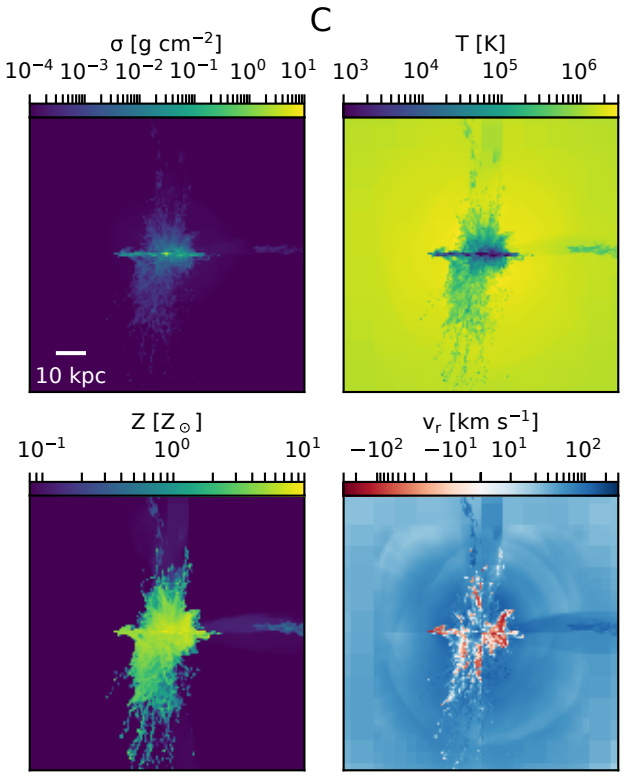


Figure 6: Shows the same quantities as Figure 4, but for Point C of Figure 3 at 7.00 Gyr. The galaxy has achieved an approximate steady-state. Recent outflows (red) are small, and inflows (dark blue) are highly collimated; both are the only enriched gas.

the same behavior as seen in Section 3.1.1 with the original fiducial models. We also see that the temperature increases over time, as before. Unlike the original run, however, the density increases slightly around the disk.

From the radial velocity projections of both Figures 9 and 10, we see highly collimated material falling onto the galaxy. Just as with the original simulation, all the material that blows out from the disk eventually falls back in. Moreover, we again see that the outflows have a lower entropy than the surrounding CGM. The metal-rich outflowing gas does not become buoyant, and the CGM remains unenriched as with the original model.

This simulation was stopped before 5 Gyr, not because of slow progress, but because of insufficient difference from the original feedback model in terms of the CGM metallicity and outflow entropy. Additionally, there were also no variations in kinetic fraction f_k tested, because for $0 < f_k < 1$, the precise value had no effect on the evolution of the CGM (see Section 3.1.2; values of 0 and 1 for f_k do have an effect but are physically unrealistic).

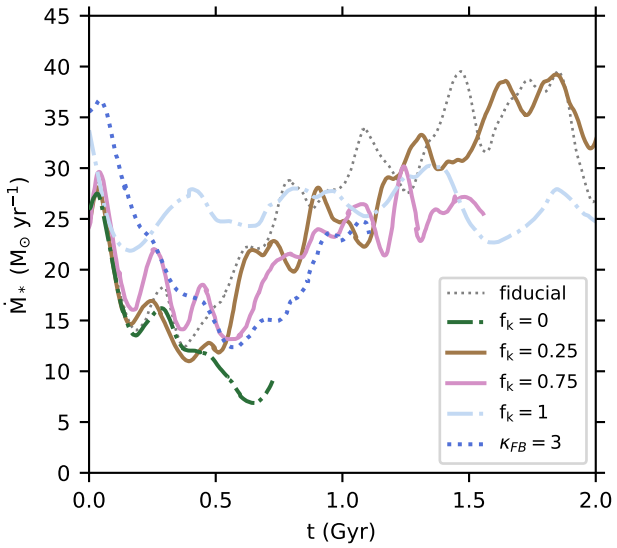


Figure 7: Star formation rate (SFR) over time for parameter variations of the original model. The fiducial simulation (Figure 3) is shown as the thin grey dotted line. Extreme variations in the kinetic fraction f_k are shown as dash-dotted lines. Values near the fiducial $f_k = 0.5$ are solid lines. The increase in feedback extent κ_{FB} is the thick dotted line. All curves have been smoothed using the same method and parameters as in Figure 3.

4 DISCUSSION

The aim of this work was to create an idealized, simplified model of star formation and feedback that does not have the same limitations as the standard particle-based star formation and feedback models typically used. Such a model was intended to explore how galaxies self-regulate themselves in a computationally efficient manner. We developed two models, an original model and a revised version, that emulated instantaneous star formation and feedback in a volumetric manner. The star formation rate of the original model was unrealistically high, which is partly why the revised model was developed. The revised model reduced the star formation rate through the removal of extra cold gas.

We see in Section 3, however, that these feedback models fail to produce realistic CGMs. A real CGM has metals [Tumlinson et al. 2017]. Both the original and revised feedback models (Sections 3.1 and 3.2) see a build-up of metal in the disk while their CGMs remain at the metallicity of the initial conditions. While the feedback is able to drive enriched outflows, this gas has insufficient entropy to achieve buoyancy in the CGM and instead falls back on to the disk. It's not entirely clear why the entropy of the ejected gas is so low. With these simulations, enough thermal energy is added to boost the temperature of the gas to 10^7 K, but the ejecta quickly fragments and cools. It may be that the gas is in a density and metallicity regime where it cools very efficiently, so that its entropy is always below that of the surrounding CGM. If metal enriched gas is not sufficiently hot, its cooling efficiency only increases as its

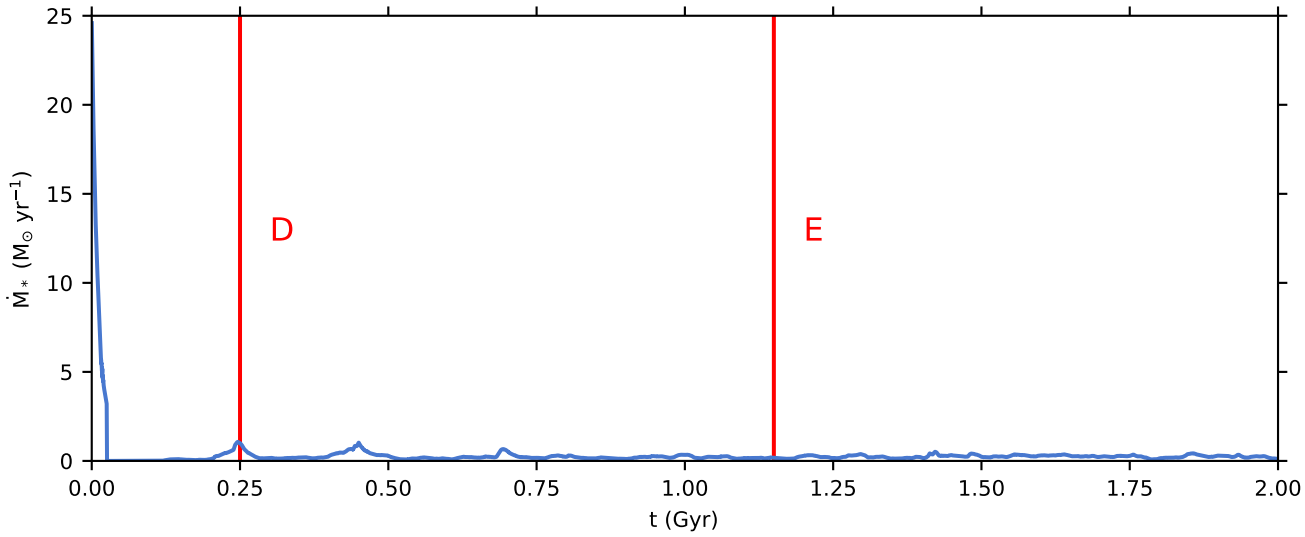


Figure 8: Star formation rate (SFR) over time for the revised fiducial run. Unlike in Figure 3, the blue line has not been smoothed. The vertical red lines labeled D and E are at 0.25 and 1.15 Gyr, and correspond to Figures 9 and 10.

temperature drops and runaway cooling occurs. The low entropy problem is also likely compounded by the continual heating of the CGM: the temperature difference between the ejecta and its surroundings is ever increasing, thereby making buoyancy an ever harder state for the ejecta to achieve.

Further reasonable modifications to the revised model would result in a method similar to an existing particle method, which is what this work was trying to avoid. The failure of both the original and revised models to produce a realistic, enriched CGM strongly suggests to us that a volumetric feedback scheme cannot replace a method that directly models stellar populations.

4.1 Student Challenges[5]

This was my first experience to both inheriting another person's code, as well as modifying an existing code base. As a result, the majority of my time at the beginning of this project was spent learning the internals of Enzo and logic of the method my predecessor had designed but never tested. As I stepped in to modifying Enzo for myself, I became much better at debugging and even had the occasion to use a memory profiler. Many of the error messages Enzo produces, as is often the case with a complex code, were symptoms of the underlying problems rather than directly related to the bug. This was how I was exposed to the many ways hydrodynamic solvers can fail, and how to mind Enzo's AMR hierarchy.

I presented this project at three academic conferences; once while it was in progress and twice when it was finished. Since this work took place during my final year of my Bachelor's degree, the experience gained from this project was a large component of my graduate school and fellowship applications. As of this writing I am a first-year graduate student with the Department of Energy Computational Science Graduate Fellowship. The work done as a part of this project has been tremendously helpful in preparing me for this fellowship.

[5]Written from the perspective of C. Kopenhafer.

5 SUMMARY

This work sought to create a volumetric model for star formation and feedback that was more computationally efficient than existing particle-based methods. Our goal was to use this model to explore how galaxies self-regulate themselves by examining the bulk properties of isolated galaxy simulations. Unfortunately, our volumetric feedback models failed to produce realistic galaxies:

- In our original model, the star formation rate was unrealistically high for a Milky Way-like galaxy.
- Additionally, the feedback failed to enrich the CGM with metals like in a real galaxy, because the ejecta did not become buoyant.
- The revised model fixed the SFR problem, but not the enrichment problem; ejecta still failed to become buoyant.
- Metal-enriched, ejected gas is likely cooling too efficiently to attain buoyancy.

Any further modifications we considered moved the models closer to the existing computationally-expensive particle methods we were trying to avoid. We therefore found that our volumetric models are not a plausible way of treating star formation and feedback in isolated galaxies. A different approach may still yield a model that avoids the expense of particles, and allow for the efficient examination of bulk galaxy properties.

6 ACKNOWLEDGEMENTS

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois, using NSF PRAC ACI-1514580. This research was supported by NASA

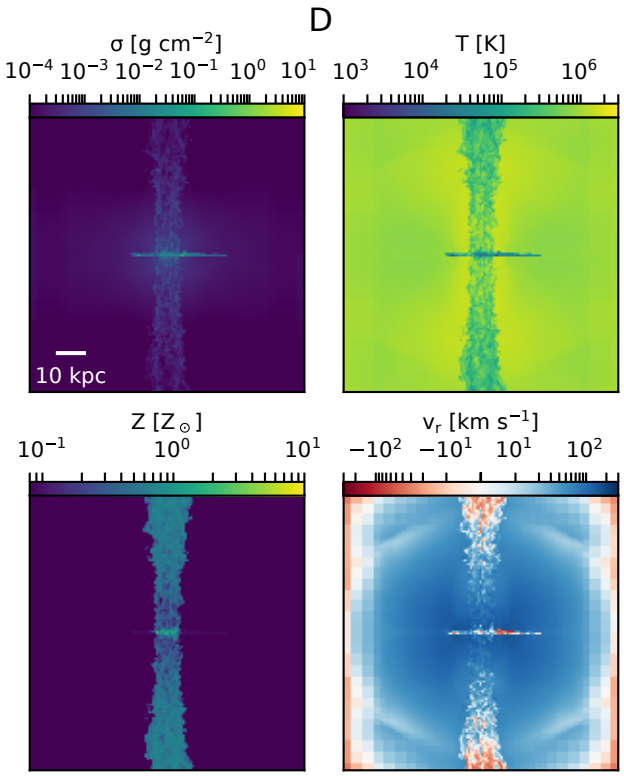


Figure 9: Projections of the revised fiducial run, in the same quantities as Figure 4, but for Point D of Figure 8 at 0.25 Gyr. The galaxy is shown during an early burst of star formation, with part of an earlier metal-enriched outflow beginning to fall back on to the galaxy (blue).

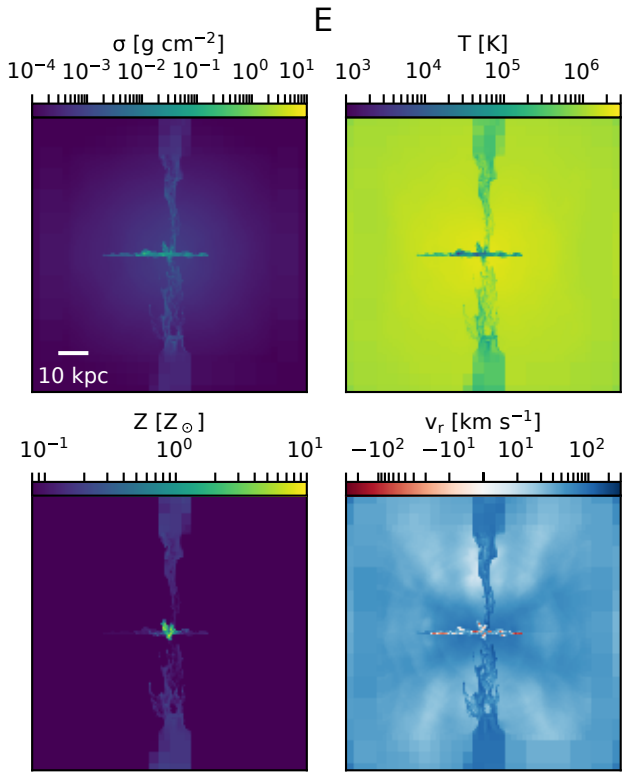


Figure 10: Same quantities as Figure 9, but for Point E of Figure 8 at 1.15 Gyr. The galaxy is in a quiescent period with no large-scale outflows; however, there are highly collimated inflows of enriched gas (dark blue).

grants NNX12AC98G, HST-AR-13261.01-A and HST-AR-13895.001. This work was performed using the open-source Enzo and yt codes, which are the products of collaborative efforts of many independent scientists from institutions around the world. Their commitment to open science has helped make this work possible.

C. Kopenhafer would also to acknowledge Shodor and their Blue Waters Student Internship Program for supporting this project and providing training in high performance computing; Greg Meece for writing the code from which this project was built; and The Department of Energy Computational Science Graduate Fellowship, for providing support while this paper was being written.

REFERENCES

Greg L. Bryan, Michael L. Norman, Brian W. O'Shea, Tom Abel, John H. Wise, Matthew J. Turk, Daniel R. Reynolds, David C. Collins, Peng Wang, Samuel W. Skillman, Britton Smith, Robert P. Harkness, James Bordner, Ji-hoon Kim, Michael Kuhlen, Hao Xu, Nathan Goldbaum, Cameron Hummels, Alexei G. Kritsuk, Elizabeth Tasker, Stephen Skory, Christine M. Simpson, Oliver Hahn, Jeffrey S. Oishi, Geoffrey C. So, Fen Zhao, Renyue Cen, and Yuan Li. 2014. ENZO: An Adaptive Mesh Refinement Code for Astrophysics. *The Astrophysical Journal Supplement, Volume 211, Issue 2, article id. 19, 52 pp. (2014).* 211 (apr 2014), 19. https://doi.org/10.1088/0067-0049/211/2/19

P. Colella and Paul R. Woodward. 1984. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics (ISSN 0021-9991), vol. 54, April 1984, p. 174-201.* 54 (sep 1984), 174–201. https://doi.org/10.1016/0021-9991(84)90143-8

Genevieve J. Graves, S. M. Faber, and Ricardo P. Schiavon. 2009. Dissecting the Red Sequence. I. Star-Formation Histories of Quiescent Galaxies: The Color-Magnitude versus the Color-σ Relation. *The Astrophysical Journal, Volume 693, Issue 1, pp. 486-506 (2009).* 693 (mar 2009), 486–506. https://doi.org/10.1088/0004-637X/693/1/486

Ji-hoon Kim, Oscar Agertz, Romain Teyssier, Michael J. Butler, Daniel Ceverino, Jun-Hwan Choi, Robert Feldmann, Ben W. Keller, Alessandro Lupi, Thomas Quinn, Yves Revaz, Spencer Wallace, Nickolay Y. Gnedin, Samuel N. Leitner, Sijing Shen, Britton D. Smith, Robert Thompson, Matthew J. Turk, Tom Abel, Kenza S. Arraki, Samantha M. Benincasa, Sukanya Chakrabarti, Colin DeGraf, Avishai Dekel, Nathan J. Goldbaum, Philip F. Hopkins, Cameron B. Hummels, Anatoly Klypin, Hui Li, Piero Madau, Nir Mandelker, Lucio Mayer, Kentaro Nagamine, Sarah Nickerson, Brian W. O'Shea, Joel R. Primack, Santi Roca-Fàbrega, Vadim Semenov, Ikkoh Shimizu, Christine M. Simpson, Keita Todoroki, James W. Wadsley, John H. Wise, and AGORA Collaboration. 2016. The AGORA High-resolution Galaxy Simulations Comparison Project. II. Isolated Disk Test. *The Astrophysical Journal, Volume 833, Issue 2, article id. 202, 34 pp. (2016).* 833 (dec 2016), 202. https://doi.org/10.3847/1538-4357/833/2/202

Alan W. McConnachie. 2012. The Observed Properties of Dwarf Galaxies in and around the Local Group. *The Astronomical Journal, Volume 144, Issue 1, article id. 4, 36 pp. (2012).* 144 (jul 2012), 4. https://doi.org/10.1088/0004-6256/144/1/4

Stacy S. McGaugh. 2005. The Baryonic Tully-Fisher Relation of Galaxies with Extended Rotation Curves and the Stellar Mass of Rotating Galaxies. *The Astrophysical Journal, Volume 632, Issue 2, pp. 859-871.* 632 (oct 2005), 859–871. https://doi.org/10.1086/432968 arXiv:astro-ph/0506750

J. F. Navarro, C. S. Frenk, and S. D. M. White. 1996. The Structure of Cold Dark Matter Halos. *The Astrophysical Journal* 462 (may 1996), 563. https://doi.org/10.1086/177173 arXiv:astro-ph/9508025

Britton D. Smith, Greg L. Bryan, Simon C. O. Glover, Nathan J. Goldbaum, Matthew J. Turk, John Regan, John H. Wise, Hsi-Yu Schive, Tom Abel, Andrew Emerick, Brian W. O'Shea, Peter Anninos, Cameron B. Hummels, and Sadegh Khochfar. 2017. GRACKLE: a chemistry and cooling library for astrophysics. *Monthly Notices of the Royal Astronomical Society, Volume 466, Issue 2, p.2217-2234* 466 (apr 2017),

2217–2234. https://doi.org/10.1093/mnras/stw3291

James M. Stone and Michael L. Norman. 1992. ZEUS-2D: A radiation magnetohydrodynamics code for astrophysical flows in two space dimensions. I - The hydrodynamic algorithms and tests. *Astrophysical Journal Supplement Series (ISSN 0067-0049), vol. 80, no. 2, June 1992, p. 753-790. Research supported by University of Illinois.* 80 (jun 1992), 753–790. https://doi.org/10.1086/191680

Jason Tumlinson, Molly S. Peeples, and Jessica K. Werk. 2017. The Circumgalactic Medium. *Annual Review of Astronomy and Astrophysics, vol. 55, issue 1, pp. 389-432* 55 (aug 2017), 389–432. https://doi.org/10.1146/annurev-astro-091916-055240

G. Mark Voit, Greg L. Bryan, Brian W. O'Shea, and Megan Donahue. 2015a. Precipitation-regulated Star Formation in Galaxies. *The Astrophysical Journal Letters, Volume 808, Issue 1, article id. L30, 5 pp. (2015).* 808 (jul 2015), L30. https://doi.org/10.1088/2041-8205/808/1/L30

G. Mark Voit and Megan Donahue. 2015. Cooling Time, Freefall Time, and Precipitation in the Cores of ACCEPT Galaxy Clusters. *The Astrophysical Journal Letters* 799 (jan 2015), L1. https://doi.org/10.1088/2041-8205/799/1/L1

G. M. Voit, M. Donahue, G. L. Bryan, and M. McDonald. 2015b. Regulation of star formation in giant galaxies by precipitation, feedback and conduction. *Nature, Volume 519, Issue 7542, pp. 203-206 (2015).* 519 (mar 2015), 203–206. https://doi.org/10.1038/nature14167

G. Mark Voit, Megan Donahue, Brian W. O'Shea, Greg L. Bryan, Ming Sun, and Norbert Werner. 2015c. Supernova Sweeping and Black Hole Feedback in Elliptical Galaxies. *The Astrophysical Journal Letters, Volume 803, Issue 2, article id. L21, 5 pp. (2015).* 803 (apr 2015), L21. https://doi.org/10.1088/2041-8205/803/2/L21

G. Mark Voit, Greg Meece, Yuan Li, Brian W. O'Shea, Greg L. Bryan, and Megan Donahue. 2017. A Global Model for Circumgalactic and Cluster-core Precipitation. *The Astrophysical Journal, Volume 845, Issue 1, article id. 80, 27 pp. (2017).* 845, Article 80 (aug 2017), 80 pages. https://doi.org/10.3847/1538-4357/aa7d04 arXiv:1607.02212

Jessica K. Werk, J. Xavier Prochaska, Jason Tumlinson, Molly S. Peeples, Todd M. Tripp, Andrew J. Fox, Nicolas Lehner, Christopher Thom, John M. O'Meara, Amanda Brady Ford, Rongmon Bordoloi, Neal Katz, Nicolas Tejos, Benjamin D. Oppenheimer, Romeel Davé, and David H. Weinberg. 2014. The COS-Halos Survey: Physical Conditions and Baryonic Mass in the Low-redshift Circumgalactic Medium. *The Astrophysical Journal, Volume 792, Issue 1, article id. 8, 21 pp. (2014).* 792 (sep 2014), 8. https://doi.org/10.1088/0004-637X/792/1/8

April 2018

Volume 9 Issue 1

TABLE OF CONTENTS

ISSN 2153-4136 (online)