EA4 Lab 4:

11/26/19

Adolphus Adams, Preston Collins, Shane Dolan, and Braden Rocio

Certification Page

We certify that the members of the team named below worked on this labusing only our own ideas, possibly with the help from the TAs and our professors,specifically in the writing of the report, the programming and the analysis required for each task. We did not collaborate with any members of any other team nor did we discuss this lab or ask assistance from anyone outside of our team, the TAs and our professors. If we accessed any websites in working on these labs, these websites are explicitly listed in our report (with the exception of websites used only to get information on specific Matlab programs). We understand that there will be severe consequences if this certification is violated and there is unauthorized collaboration.

1. Shane Dolan -  - 11/26/19

2. Braden Rocio -  - 11/26/19

3. Adolphus Adams  - 11/26/19

4. Preston Collins -  - 11/26/19

Introduction:

In lab 4, we will investigate the difference between stiff equations and their solutions as well as the difference in precision of MATLAB's built-in numerical solvers: *ode15s* and *ode45* when used to solve stiff equations. According to MATLAB's documentation, *ode15s* is designed to solve stiff equations so we expect it to perform better when solving a system of stiff equations compared to *ode45*. In addition, we will focus on more complex behavior for nonlinear systems by observing what happens when critical points are unstable. Finally, we will use systems of differential equations to model chemical reactions.

A stiff system is one which multiple timescales are used in order to maintain the numerical stability of the solution. These systems occur quite frequently and require that a smaller timescale is used in order to ensure that the solution is precise. During this lab we will use both *ode45* and *ode15s* to solve systems. *Ode15s* lends itself to the stiff case because it is an implicit solver that allows for the manipulation of timestep size in order to achieve an error tolerance. This type of solver differs from an explicit solver that which would require a very small timestep in order to achieve the same level of precision. We will compare the number of time steps required to solve an equation with each of these solvers.

Part 1

Task 1:

Introduction:

In Task 1.1, we will take a first order system of linear equations and use MATLAB to find the exact solution using the method of eigenvalues. We will also use the initial conditions provided in the lab documentation in order to solve the initial value problem. The solutions for each equation will be plotted in order to analyze the transient and long term behavior of the solution over time.

Task 1.1

MATLAB comes with functions that can solve for both eigenvalues and eigen vectors. We will take advantage of these convenient functions and solve for the eigenvectors for the equation:

$$\overline{X'} = A\overline{X}$$

For A:

$$A = [96\ 2\ -99;\ 101\ -3\ -99;\ 196\ 2\ -199]$$

And

$$\overline{X}(0) = [2;\ -2;\ 1]$$

Such that the solution X(t) is in the form

$$\overline{X}(t) = [x(t);\ y(t);\ z(t)]$$

And particularly, the longe term behavior is governed by the terms

$$\overline{XL}(t) = [xl(t);\ yl(t);\ zl(t)]$$

MATLAB can solve for the normalized eigenvectors using *eig*.

The unscaled eigenvectors computed by matlab for the system above are:

$$\lambda = -100, -1, -5$$

We know the standard form of the solution for a system of this order with real eigenvalues is:

$$X(t) = c_1 e^{-\lambda_1 t}\widehat{v_{\lambda_1}} + c_3 e^{-\lambda_3 t}\widehat{v_{\lambda_3}} + c_3 e^{-\lambda_3 t}\widehat{v_{\lambda_3}}$$

Where $V_{\lambda i}$ is an eigenvector for the matrix A.

Next we solve the initial value problem for the constants C_i by evaluating:

$$\overline{C} = A \backslash \overline{b}$$

For *b:*

$$b = \overline{X_0}$$

Now we can substitute all known values into our solution and break it into 3 separate equations allowing us to solve each component of the solution at each point in time that we want. We will evaluate each component from 0<t<6 with a timeste of 0.01. This produces an array of the exact solution at each point t computed. We can now plot each solution individually as a function of time and analyze the behavior.

We know that the long term behavior for each component is governed by the term with |λ| closest to 0 because an exponential function defined by this parameter decays most slowly. In order to asses the transient and long term behavior of each term with each component we take a closer look at the solution equations of *X*. For each component, we see that the term governing the long term behavior is clearly the term correlating to λ=-1. The terms for long term behavior of the solution are:

$$xl(t) = yl(t) = zl(t) = c_2 v_\lambda (1,2) * e^{-\lambda_1 t} = 1 * 1 * e^{-t} = e^{-t}$$

On our plots, this appears as a decaying exponential function originating at (0,1) and decaying towards *X*=0 at *t* approaches ∞.

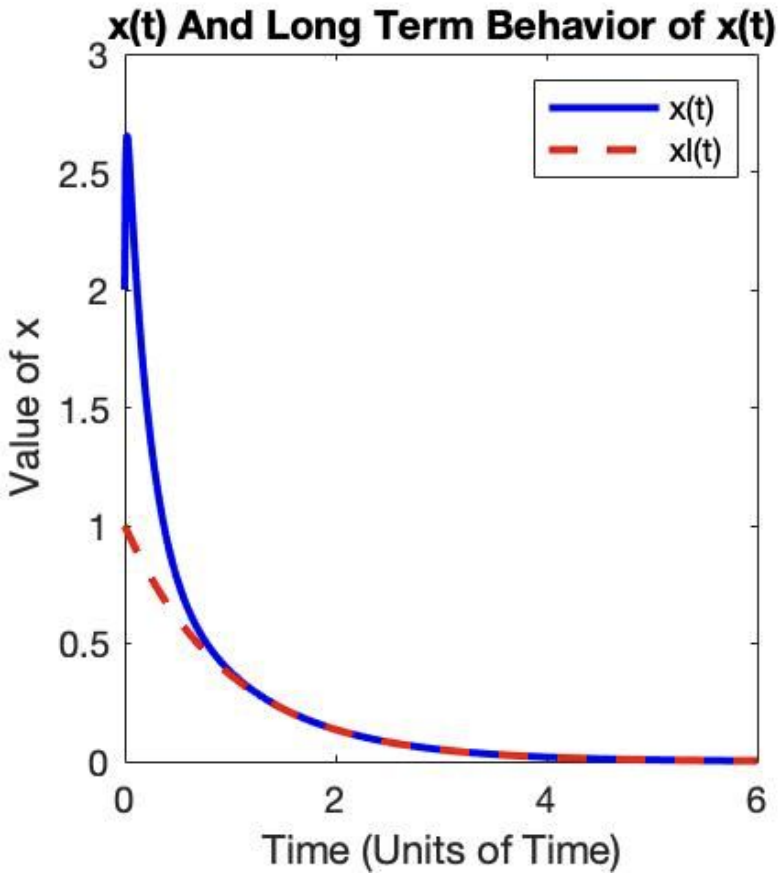

Figure 1a: *Long term behavior of x(t) as well x(t) (long term and transient behavior)*

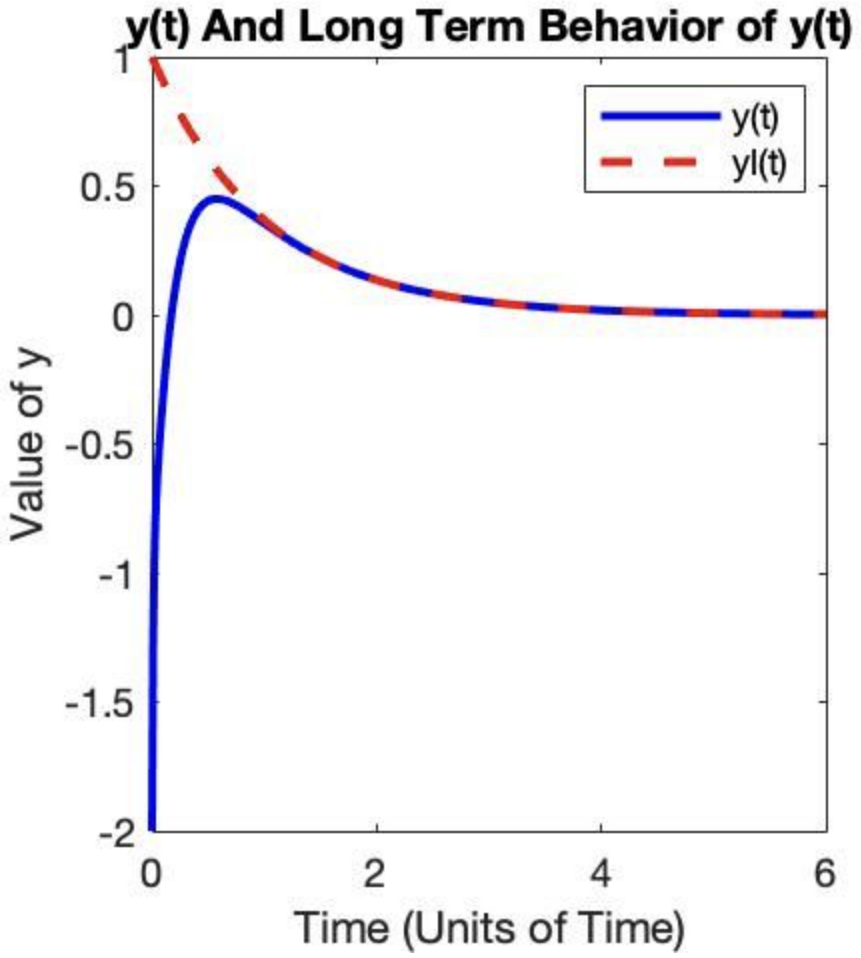

Figure 1b: *Long term behavior of y(t) as well y(t) (long term and transient behavior)*

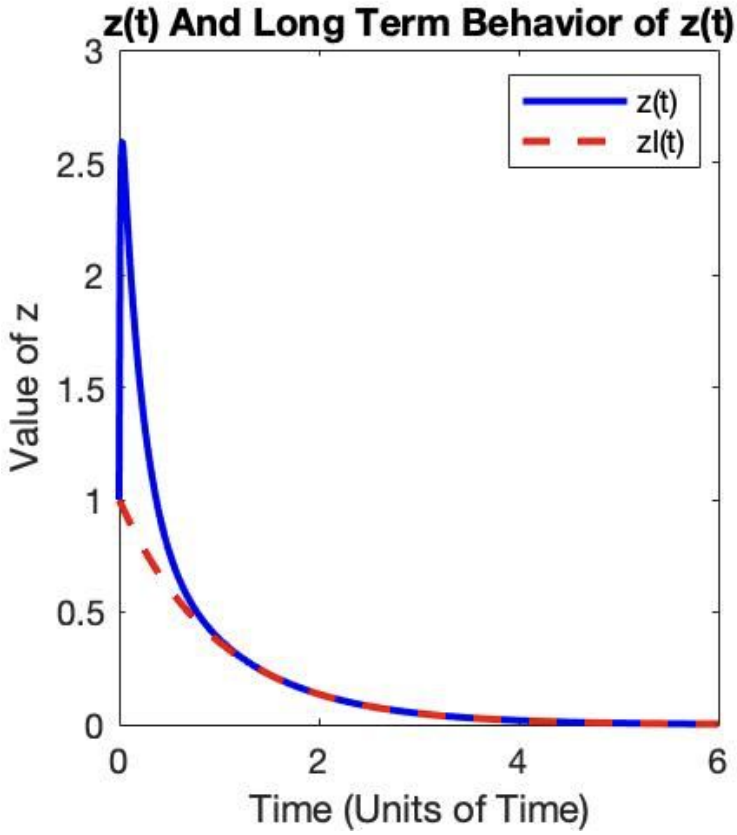

Figure 1c: *Long term behavior of z(t) as well z(t) (long term and transient behavior)*

Analysis:

In task 3, we see the behavior of each component of the solution *X*. By breaking up each component into transient and long term parts, we are able to see the impact of the transient terms on the solution of *x(t), y(t), and z(t)*. From our analysis, we see that the long term for each solution equation in the vector *X* is identical and governed by e^{-t} as *t* goes to infinity. In each of the above figures, we see that the long term behavior of the solution give the overall component's solution a characteristic shape. We also see that the overall shape of the solution (x(t) for example) matches the shape of the long term solution as T gets larger for each solution. This shows that the effect of transient terms in each component solution becomes more and more negligible as *t* increases.

By looking at the plots for each solution component, we see that the exact solution has not reached steady state at t=2s. Therefore, this would be a good location to asses the precision of a numerical scheme for the system we are analyzing. By plugging *t*=2 into our exact solution, we find that:

$$x(t = 2) = 0.1354,\ y(t = 2) = 0.1352,\ z(t = 2) = 0.1354$$

Task 2:

Introduction:

In Task 1.2, we use *ode15s* and *ode45* to numerically solve for the solution to the systems of equations above. The first line of MATLAB's documentation for *ode45* states "*Solve non-stiff differential equations"* and the documentation for *ode15s* says *"Solve stiff differential equations."* We predict that *ode15s* will likely lend itself to solving the stiff equation in this lab more efficiently that *ode45*. In order to measure the efficiency of each solver, we will first compare the results at *t*=2 to the exact solution from the previous task to ensure that our solution is precise. Next, we will analyze the number of times *prime.m* is called (*count*) and analytically associate this with the number of steps (*nsteps*) required to achieve the level of precision needed to model the exact solution. We accumulate the number of times *prime.m* is called in the variable *count* and the number of time steps used to arrive at our solution in *nsteps*. Global variables are defined for *A* and *count* to simplify the program and keep track of the number of iterations. *Count* is set to 0 at the beginning of each solution attempt. The results for each component of *X* are plotted on the same axis to ensure the numerical solutions resemble the exact solution computed in Task 1.1. The results of each computation at t=2s is found in the chart below. Plots for the numerically computed solutions, *X,* are displayed below for both *ode45stiff.m* and *ode15sstiff.m*.

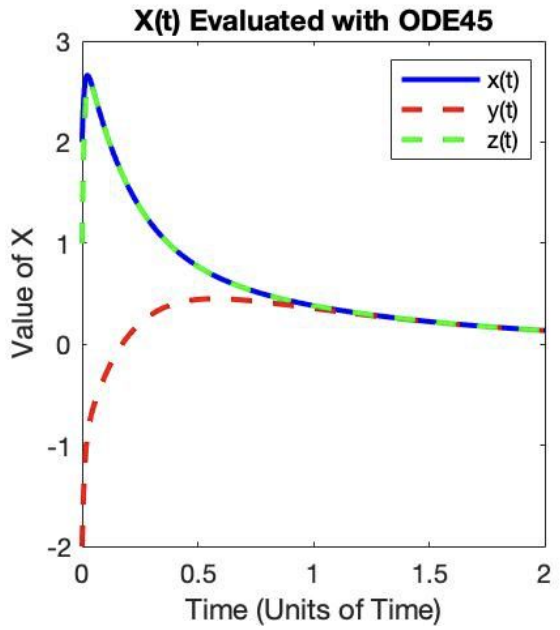

Figure 2: *Components of X(t) for 0<t<2 solved use ode45 in ode45stiff.m*

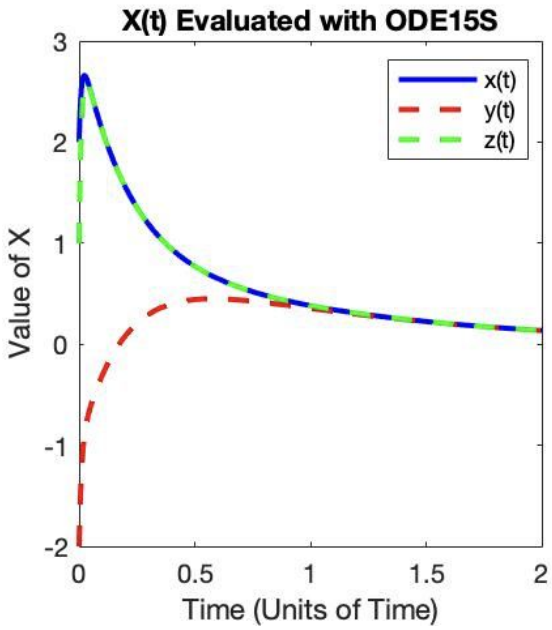

Figure 3: *Components of X(t) for 0<t<2 solved use ode15s in ode15sstiff.m*

Analysis:

In Task 1.2 we the precision of *ode15s* is compared to *ode45*. Based on our numerical soluti n we see that the solutions produced by both numerical methods are nearly exactly the same as the exact solution solved in Task 1.1. We do see however, that there is a large difference between the number of time *prime.m* is called and the number of time steps required to arrive at the solution for *ode15s* and *ode45*. The results of each solution are shown below:

Method	***nsteps***	***count***
ode45	281	151
ode15s	72	421

Task 3:

Introduction:

In task 3, we will compare the efficiency of each solution method used above in order to make more informed decisions regarding which solver to use in the future. In the previous task, we programed two accumulators that summed the total number of times *yprime.m* was called and the number to timesteps used to compute the above solutions (*nsteps*). In this task we will extend the analysis of numerical efficiency by assessing the error of each solution and interpreting each metric.

Task 3

The lab manual calls us to define the error in each solution as the maximum absolute difference between the exact solution and each numerical solution at t=2. That is, for example, the largest difference between a component of X_{exavt} and X as computed by *ode15sstiff.m.* When doing our analysis, it is important to ensure that the error between each numerical solution is roughly the same such that we can compare *ode15s* and *ode45* to each other under controlled conditions. In MATLAB, this is:

error=max(max((abs(X2exact-X2s))

We will also compute the ratio between *nsteps* and *count* for both numerical solutions. The results of the above summary are displayed in the following plot:

N15s	***C15s***	***N45***	***C45***	***RN***	***RC***	***e15***	***e45***
72	151	281	421	0.256	0.359	1.7826e-04	1.8565e-04

Table 1: *Analytics on number of timesteps and number of times prime.m called in order to achieve a max error tolerance on the order or 10^{-4} compared to the exact solution for both ode45 and ode15s.*

Analysis:

1. First, it should be noted that the error between the exact solution and the numerical solutions for each numerical scheme is very similar and on the same order of magnitude for t=2. This ensures the validity of comparisons made as the solvers are operating under similar conditions.
2. In task 1 we see that the number of time steps required to solve the initial value problem for an error tolerance on the order of 10^{-4} is much higher for *ode45* compared to *ode15s*. We can extrapolate this metric and conclude that the step size required to reach this tolerance is much smaller for *ode45* because our solver duration is finite and identical for each method used. This can be attributed to the fact that *ode15s* is an implicit method meaning that the solution is found by solving an equation using information from the current state and later state. An explicit method, like *ode45*, calculates the solution at a

late time using only the information available at the current state. Implicit methods allow for larger time steps than explicit and are more likely to be unconditionally stable if a large enough time step is able to be used, this situation isn't possible to occur in an explicit scheme such as *ode45*. The ratio between *n15s* and *n45* is *RN*=0.256 indicating that it took *ode15s* 25% of the steps it took *ode45* to solve the solution for a similar max error.

3. Assuming that the number of time *prime.m* is called (*count*) is a measure of the computational power required to arrive at each solutions. We see that it took *ode15s* significantly less computational power to solve the system. From the ratio *RC* computed above, we see that it took *ode15s* 36% of the computational power it took *ode45* to reach an error threshold on the same order of magnitude.

Task 4:

Introduction:

The purpose of task 4 is to create a report that explains the results of the tests done in tasks 1.1-1.3.

Analysis:

In Task 1.1 we found the exact solution to the system of differential equations with MATLAB. We used the *eig* function to find the normalized eigenvalues and eigenvectors for A, put the equation into standard for, solved the coefficients for the initial values problem and then found the exact solution for X as a function of time and plotted each component. Moreover, we computed the exact value of X at t=2. In Task 1.1. Specifically in figures 1a-c we see the relationship between the long-term governing term for each component of X and the over all solution for that component. In particular, as mentioned previously, we can see that as t approaches infinity, the overall solution for that term is equal to the long term behavior that is plotted in the same figure. This highlights the fact that the transient is disappearing faster than the long term parts of the solution. The value of the transient terms can also be derived from our plots by finding the difference between the long term solution and the overall solution for each component.

In Task 1.2 we created two separate programs that would numerically solve the initial value problem using MATLAB's built in functions *ode45* and *ode15s*. The numerical solutions using *ode45* and *ode15s* were plotted in figures 2 and 3. It is clear that each component looks nearly identical to the exact solutions plotted in 1a-c. Graphically, it does not appear that there is any difference between the two numerical solutions. Both programs were instructed to accumulate the number of times *prime.m* were counted in order to measure the computational efficiency of each. We also recorded the number of time steps for our [0,1] timespan. This allowed us to calculate the size of the time step required to reach an error threshold that we were intending. In order to compare these two different numerical solvers, we first ensure that the error between the numerically computed solution and the exact solution at t=2 is approximately the same for each of the solutions. These two errors are tabulated in the lat two columns of Table 1. After we computed two similar solutions we were able to compare the results of each solution. In Task 1.3 we see that *ode15s* was far more efficient at solving the stiff system. From the ratio *RC* we see that it took *ode15s* 36% of the number of computations it took *ode45* (the number of times *prime.m* is called). Likewise, we see that the average time step used in *ode15sstiff.m* used 26% of the number of timesteps compared to *ode45*. This indicates that *ode15s* was able to accomplish a similar level of error with a much larger time step.

From Tasks 1.1-1.3 we are drawn to conclude that if a numerical method is required, one should use *ode15s* in order to solve a system of stiff equations. If it is practical, it is not very challenging to create code that will some the stiff system exactly using eigenvalues. We know this decision is

validated by the average time step size and count variables computed in *ode15sstiff.m* and *ode45.m* as well as the similar maximum error computed for each numerical solution.

Part 2

Task 1:

Brussels, Belgium

Task 2:

Introduction:

The purpose of task 2 is to take the given Y' and X' equations and solve for the critical point of the function in terms of k_1, k_2, k_3, k_4, A, and B.

Analysis:

$$\textbf{6a: } \frac{dX}{dt} = k_1 A - k_2 BX + k_3 X^2 Y - k_4 X$$

$$\textbf{6b: } \frac{dY}{dt} = k_2 BX - k_3 X^2 Y$$

Taking these two equations we can set them both to 0 and solve for X and Y to find the critical points.

$$0 = k_1 A - k_2 BX + k_3 X^2 Y - k_4 X$$

$$0 = k_2 BX - k_3 X^2 Y$$

From here we can solve for Y in terms of X and then plug it into the first equation to get X.

$$0 = k_2 BX - k_3 X^2 Y$$

$$k_3 X^2 Y = k_2 BX$$

$$Y = \frac{k_2 BX}{k_3 X^2}$$

$$Y = \frac{k_2 B}{k_3 X}$$

Then we plug this in for Y into the first equation:

$$0 = k_1 A - k_2 BX + k_3 X^2 Y - k_4 X$$

$$0 = k_1 A - k_2 BX + k_2 BX - k_4 X$$

$$X = \frac{k_1 A}{k_4}$$

We then can plug this back into Y to solve for Y

$$Y = \frac{k_2 B}{k_3 X}$$

$$Y = \frac{k_2 B k_4}{k_3 k_1 A}$$

The critical point for this function is at: $\left(\frac{k_1 A}{k_4} , \frac{k_2 B k_4}{k_3 k_1 A} \right)$ and is always positive because k_1, k_2, k_3, k_4, A, and B are all positive.

Task 3:

Introduction:

In Task 3 the equations of $\frac{dX}{dt}$ and $\frac{dY}{dt}$ are evaluated in a Jacobian at $k_2=k_3=k_4=A=1$, B=1 and then $k_1=1.5$ and then $k_1=1.4$ in order to prove that at $k_1=1.5$ the solution is a stable spiral and at $k_1=1.4$ the solution is an unstable spiral. We will use the equations given to us in class to determine whether or not it is stable. These equations are:

$\lambda = -a \pm ib$ Stable

$\lambda = a \pm ib$ Unstable

Analysis:

We first solve for the jacobian in terms of k_1, k_2, k_3, k_4, A, and B.

$$J = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}$$

$$J = \begin{bmatrix} -k_2B + 2k_3XY - k_4 & k_3X^2 \\ k_2B - 2k_3XY & -k_3X^2 \end{bmatrix}$$

With this we can then plug in the values we are given for k_1, k_2, k_3, k_4, A, and B.

$k_1 = k_2 = k_3 = k_4 = A = 1$ $B = 3$

$$J = \begin{bmatrix} 2 & {k_1}^2 \\ -3 & -{k_1}^2 \end{bmatrix}$$

From here we can plug in the two given k_1 values of 1.4 and 1.5 which gives:

$k_1=1.5$

$\lambda^2 + .25\lambda + 2.25 = 0$

And when put through the quadratic equation provides us with the eigenvalues

$$\lambda = -.125 \pm \frac{\sqrt{-8.9375}}{2}$$

We were taught that when the a values (-.125 in this case) is negative then the spiral is stable. On the other hand when using $k_1=1.4$ we get

$$\lambda = .02 \pm \frac{\sqrt{-1.9584}}{2}$$

Where the a value (.02 in this case) is positive meaning we have an unstable spiral.

Task 4:

Introduction:

In Task 4 we were tasked with finding the values k_{1*} where $1.4 < k_{1*} < 1.5$ so that the real part of the eigenvalues changes from negative at $k_1 > k_{1*}$ to positive when $k_1 < k_{1*}$. This will be the transition point.

Analysis:

Knowing that there's a value for k1 between 1.4 and 1.5 where the critical point for this system transitions from being an unstable spiral to a stable spiral (k_{1*}), and knowing that at this value of k1 the eigenvalues become purely imaginary, we can set the trace of the Jacobian equal to 0 (since this is equivalent to the sum of the real part of our eigenvalues being set equal to 0) and solve for k_{1*}. Setting the trace of the Jacobian equal to 0 we get…

$$2 - k1^2 = 0 \;\Rightarrow\; k1 = \sqrt{2} = 1.4142$$

Task 5:

Introduction:

In Task 5 we were asked to graph different values of k_1 in the equations $\frac{dX}{dt} = k_1 A - k_2 BX + k_3 X^2 Y - k_4 X$ and $\frac{dY}{dt} = k_2 BX - k_3 X^2 Y$ in order to see the effects of this parameter on both the solution of the differential equations (the graphs in figure 4) as well as the phase diagrams of X and Y. In addition, we will graphically compare the results of the solution produced first using the matlab solver *ode15s* and then *ode45*. We will then compare the average time step size for each as well as the computational efficiency implied by the number of times *prime2.m* is called.

Task 5:

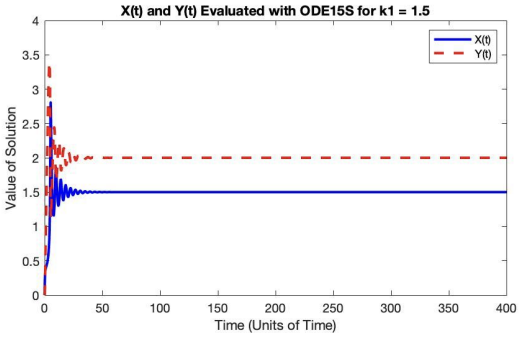

Figure 4a (above): *X(t) and Y(t) for k1=1.5*

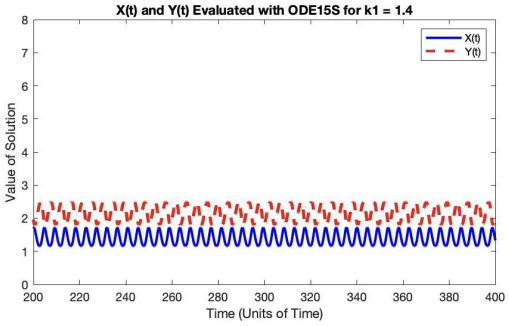

Figure 4b (above): *X(t) and Y(t) for k1=1.4*

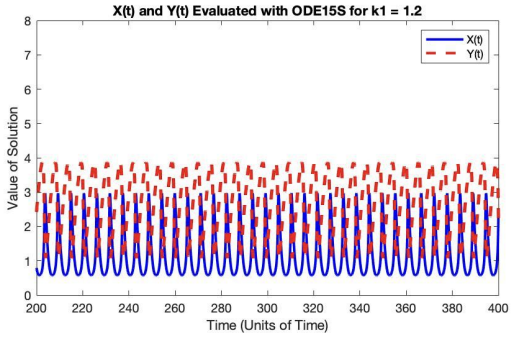

Figure 4c (above): *X(t) and Y(t) for k1=1.2*

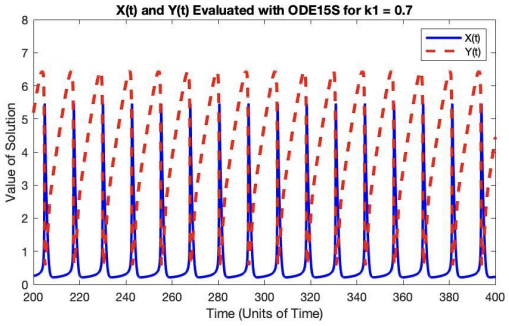

Figure 4d (above): *X(t) and Y(t) for k1=0.7*

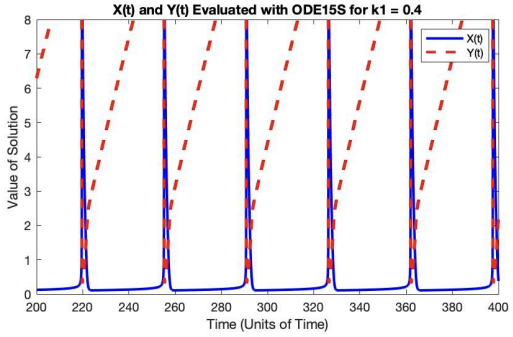

Figure 4e (above): *X(t) and Y(t) for k1=0.4*

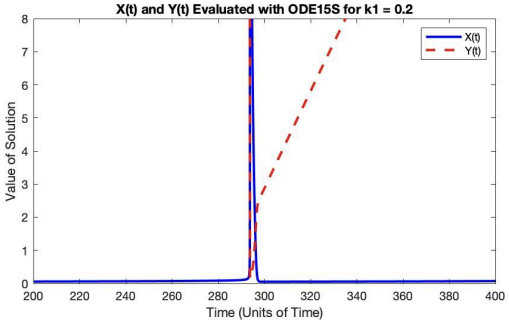

Figure 4f (above): *X(t) and Y(t) for k1=0.2*

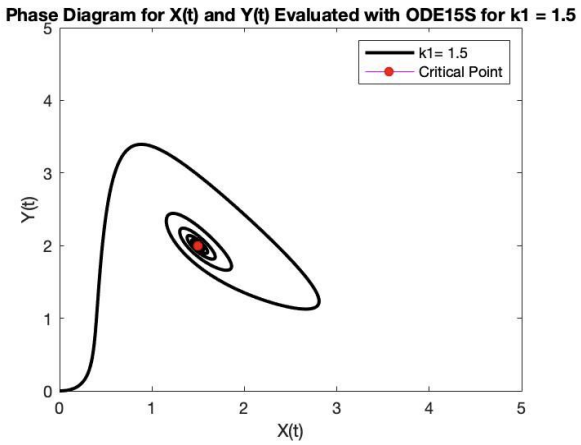

Figure 5a (above): *Phase diagram for X(t) and Y(t) with k1=1.5*

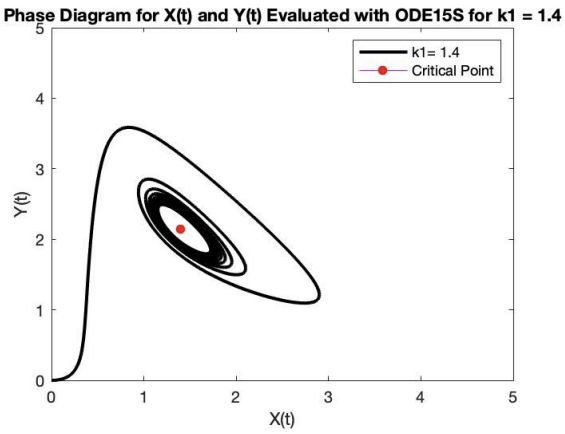

Figure 5b (above): *Phase diagram for X(t) and Y(t) with k1=1.4*

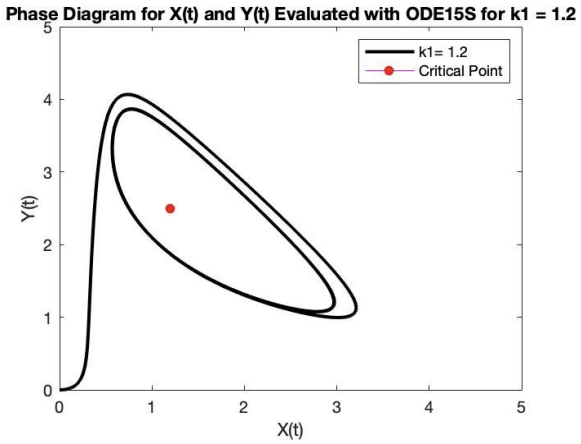

Figure 5c (above): *Phase diagram for X(t) and Y(t) with k1=1.2*

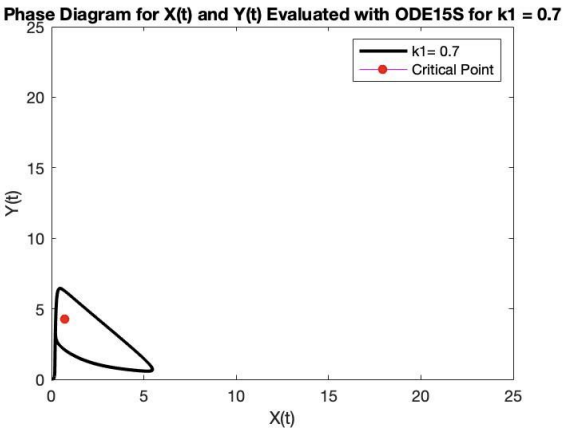

Figure 5d (above): *Phase diagram for X(t) and Y(t) with k1=0.7*

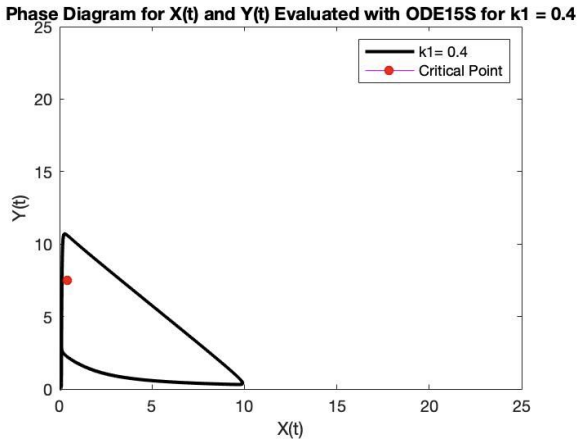

Figure 5e (above): *Phase diagram for X(t) and Y(t) with k1=0.4*

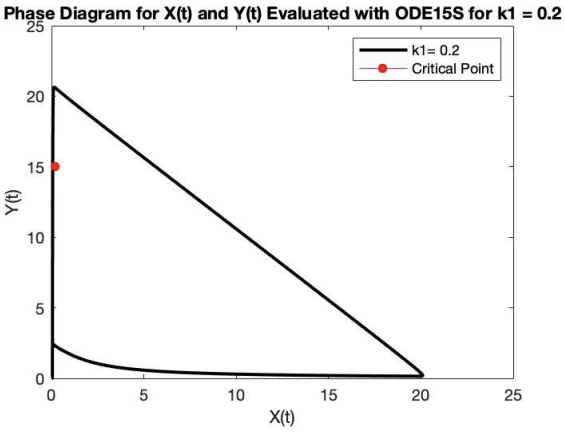

Figure 5f (above): *Phase diagram for X(t) and Y(t) with k1=0.2*

k1	N15s	C15s	N45	C45	RN	RC
1.5	1117	1705	1757	2677	0.636	0.637
1.4	5588	11259	8281	12475	0.675	0.903
1.2	9299	16608	11653	19597	0.798	0.847
0.7	7452	14517	11669	18631	0.639	0.779
0.4	3693	7830	7441	11809	0.496	0.663
0.2	945	2022	3269	5119	0.289	0.395

Table 2: *Number of time steps and values of count for ode15sbruss.m and ode45bruss.m for varying values of k.*

Analysis:

Taking a look at equation 6a and 6b we can see that the equation for 6b is inside 6a but negative. The Y peak represents when dy/dt is equal to zero and the negative of this would still be 0. This means that the k_1A-k_4X part of the 6a equation is still left behind. k_1A is a constant so it would only peak at the same time if k4X were equal to k1A when 6b is equal to 0. This can help explain why Y peaks a little earlier than X.

The key outcome of this task is to show how stiffness affects the efficiency in solving problems, so let's first look at figures 4 and 5 and discuss how stiffness is displayed. In the introduction of the lab, stiff problems were characterized by solutions that require small time steps (for accuracy when the solution is changing rapidly) in certain parts of the solution where the time scale must be small, but in other parts the solution doesn't change very much so it doesn't require the small time steps that are utilized in the rapidly changing sections. You can see in figures 4 and 5 that as the value for k1 decreases, the solutions' curvature occurs more and more rapidly, which causes a smaller time step to be needed to accurately compute the results of the curve. The rest of the solution, however, becomes more and more linear, which allows for a larger time step to be used to compute accurate results. So, in summary the solution becomes more and more stiff as k1 decreases by the characterization for stiffness given in the introduction of the lab. Looking at equation 6a, the decrease in k1 causes the change in X ($\frac{dX}{dt}$) to decrease as well which causes the solutions' curvature to change at a faster rate and makes the problem more stiff. These results are supported by the calculation made in Table 2 when comparing the use of the ode15s operator to the ode45. First, looking at the N15s and N45, we can see that in every scenario N15s is smaller than N45 which shows that ode15s is more efficient than ode45 in stiff calculations, since it's

only limited by accuracy rather than accuracy and stability (allowing the time step size to increase and decrease depending on the nature of the function), while producing relatively the same results. Also, a trend can be seen that as k1 decreases, RN decreases (ratio of N15s to N45), which means that as k1 decreases, the ode15s operator becomes more and more efficient, which makes sense since the equation becomes more and more stiff. Second, looking at C15s and C45, we see similar results where C15s is always smaller than C45 and RC generally decreases as k1 decreases. So, in summary, Table 2 proves how as the equation becomes stiffer (i.e. k1 decreases), the ode15s operator becomes more efficient when compared to the ode45 through requiring less computations to yield the same results, and the table proves this quantitatively by showing that RN and RC generally decrease as k1 decreases.

Task 6:

Introduction:

In Task 6 we were tasked with plotting 6a and 6b once again, however, this time with a fixed k_1 value of 1.2 and with multiple initial conditions. This is with the hope of seeing a curve called a limit cycle on the phase diagram

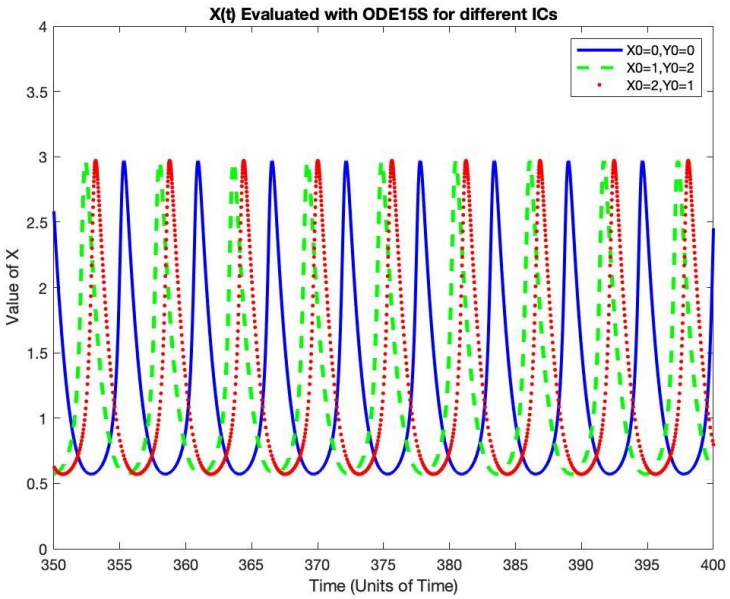

Figure 6a: *Graph of X(t): X_0=0,Y_0=0 (Blue), X_0=1,Y_0=2 (Green), X_0=2, Y_0=1 (Red)*

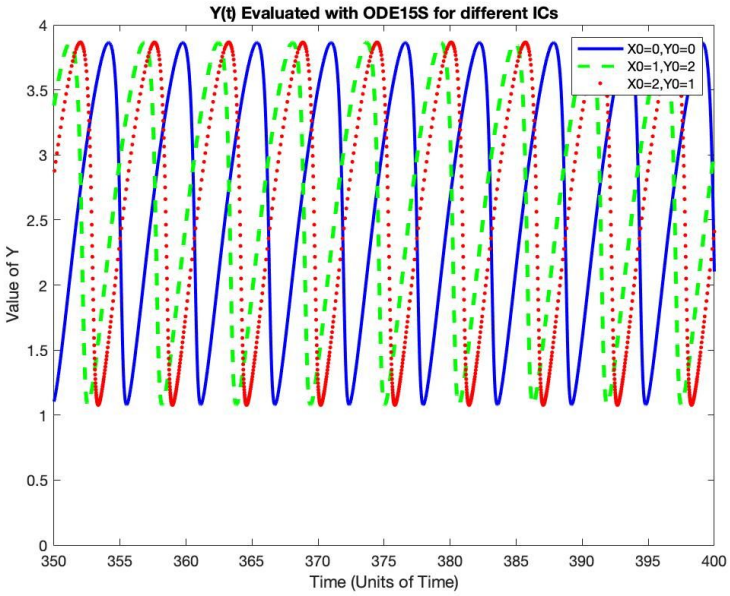

Figure 6b: *Graph of Y(t): X_0=0,Y_0=0 (Blue), X_0=1,Y_0=2 (Green), X_0=2, Y_0=1 (Red)*

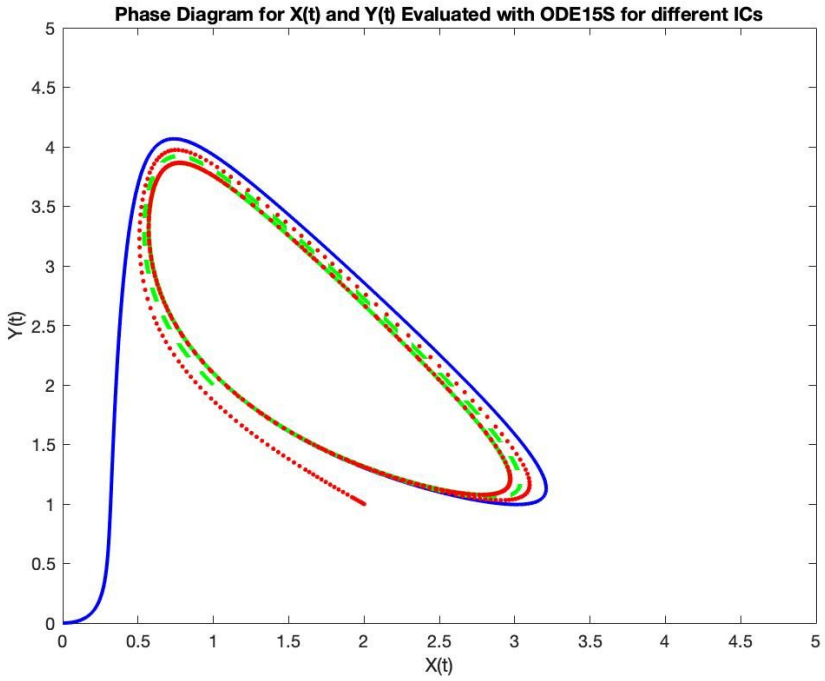

Figure 7: *Phase Diagram with initial conditions; X_0=0,Y_0=0 (Blue), X_0=1,Y_0=2 (Green), X_0=2, Y_0=1 (Red)*

Analysis:

The differences in the curves starting with different initial conditions of X(t) and the Y(t) in figure 6a and 6b is largely just that they are offset from each other. They are virtually the same outside of that, having the same amplitude and frequency. The term limit cycle accurately describes the behavior in the phase space as it sets a limit on the distance from the center point that a spiral can get. The phrase "the limit cycle in phase space contains infinitely many different solutions of (6)" means that any initial conditions given will result in the same exact limit cycle, as seen in figure 7. All of these solutions would end up at the limit cycle like the 3 plots of initial conditions in figure 7. This differs from a center as a limit cycle is a single loop around a center point that is similar to the shape of a spiral whereas a center is multiple disconnected circles around a spiral. When plotting the same data for the k_1=1.5 it appears that no limit cycle exists and it will just infinitely loop to the center point. When looking at $k_1=k_{1*}$ (as calculated in Task 4) we find a Hopf bifurcation point where the solution went from unstable at $k_1<k_{1*}$ to stable at $k_1>k_{1*}$. In both the duffing oscillator in Lab 3 and our x and y plots in figures 6a and 6b once the transient has decayed the graphs go to an oscillatory state. In Lab 4 our graphs don't have the same slopes when at the peaks and troughs whereas in Lab 3 the peaks and troughs looked identical. Although the initial conditions in Lab 3 can be said to be limited by the forced response similar to how the solution in Task 6 has its limit cycle, we can't say a bifurcation point occurred in Lab 3 because the solution for the forced oscillator is stable regardless of its initial conditions unlike the problem in Lab 4.

Conclusion & Summary

1. In Task 1 we discovered that the Brusselator was developed in the city of Brussels, Belgium.
2. In Task 2 we used algebra to solve for the critical point of the function in terms of k_1, k_2, k_3, k_4, A, and B.
3. In Task 3 we first determined the jacobian of the function in terms of k_1, k_2, k_3, k_4, A, and B. We then plugged in the given values for k_1, k_2, k_3, k_4, A, and B and showed that with varying k_1 between 1.4 and 1.5 gives either a stable spiral or unstable spiral.
4. In Task 4 we solved for the value of k_1 in between 1.4 and 1.5 where the transition point between a stable and unstable spiral was.
5. In Task 5, we used ode15s and ode45 to solve for the concentrations of the X and Y compounds within its system, and plotted their curves with respect to time and with respect to each other. We discovered that decreasing the value for k1 caused the solution to become more stiff which affected the efficiencies of these two solvers, making the ode15s solver more efficient then the ode45 when the solution became more stiff, and this was proven by looking at the decrease in the values for RC and RN as k1 decreased..
6. In Task 6 we analyzed the limit cycle when k_1=1.2 and how this doesn't exist at the point or greater than k_{1*}. We then compared the oscillators of Task 6 to the duffing oscillators in Lab 3 and the lack of bifurcation points in Lab 3 while this task contained a Hopf bifurcation point at $k_1=k_{1*}$ where the solution goes from stable to unstable.

Appendix 1: Code for Tasks 1.1,1.2,1.3

exact.m

```
global A
A=[96 2 -99
  101 -3 -99
  196 2 -199];
[V,D]=eig(A);   %compute eigen vectors of A
Scld=zeros(3);  %scale eigen vectors assuming norm 1
for i=1:3
      Scld(i,1)=V(i,1)/V(1,1);
      Scld(i,2)=V(i,2)/V(1,2);
      Scld(i,3)=V(i,3)/V(1,3);
end
t=0:.01:6;  %creat plot resolution
x=zeros(length(t),1);   %preallocate arrays for solution sets
y=zeros(length(t),1);
z=zeros(length(t),1);
xl=zeros(length(t),1);
yl=zeros(length(t),1);
zl=zeros(length(t),1);
%%
%Solve constants for initial value problem
Ai=[1,1,1;1,1,-1;2,1,1];
b=[2;-2;1];
C=Ai\b;
%%
%Find Solutions using eigenvectors and constants of the solution computed
%above.
for i=1:length(t)

x(i,1)=Scld(1,1)*exp(-100*t(i))*C(1)+Scld(1,2)*exp(-t(i))*C(2)+Scld(1,3)*exp(-5
*t(i))*C(3);

y(i,1)=Scld(2,1)*exp(-100*t(i))*C(1)+Scld(2,2)*exp(-t(i))*C(2)+Scld(2,3)*exp(-5
*t(i))*C(3);

z(i,1)=Scld(3,1)*exp(-100*t(i))*C(1)+Scld(3,2)*exp(-t(i))*C(2)+Scld(3,3)*exp(-5
*t(i))*C(3);
    xl(i,1)=Scld(1,2)*exp(-t(i))*C(2);
    yl(i,1)=Scld(2,2)*exp(-t(i))*C(2);
    zl(i,1)=Scld(3,2)*exp(-t(i))*C(2);
end
%%
%plot solutions
figure (1)
plot (t,x,'b', t,xl,'r', 'LineWidth',2.0)
xlabel('Time (Units of Time)')
```

```
ylabel('Value of x')
title('x(t) And Long Term Behavior of x(t)')
legend('x(t)','xl(t)')  %make plot look nice


figure (2)
plot (t,y,'b', t,yl,'r', 'LineWidth',2.0)
xlabel('Time (Units of Time)')
ylabel('Value of y')
title('y(t) And Long Term Behavior of y(t)')
legend('y(t)','yl(t)')  %make plot look nice

figure (3)
plot (t,z,'b', t,zl,'r', 'LineWidth',2.0)
xlabel('Time (Units of Time)')
ylabel('Value of z')
title('z(t) And Long Term Behavior of z(t)')
legend('z(t)','zl(t)')  %make plot look nice

%%
%compute X(t=2s)
i=2; %set time 2s
x2=Scld(1,1)*exp(-100*(i))*C(1)+Scld(1,2)*exp(-(i))*C(2)+Scld(1,3)*exp(-5*(i))*
C(3);
y2=Scld(2,1)*exp(-100*(i))*C(1)+Scld(2,2)*exp(-(i))*C(2)+Scld(2,3)*exp(-5*(i))*
C(3);
z2=Scld(3,1)*exp(-100*(i))*C(1)+Scld(3,2)*exp(-(i))*C(2)+Scld(3,3)*exp(-5*(i))*
C(3);
X2exact=[x2;y2;z2] %concatenate components into single array
```

Prime.m

```
%PRIME.M
function dxdt = yprime(t,x)
global count %initialize globals
global A
dxdt = A*[x(1);x(2);x(3)]; %define function as system
count=count+1; %increment global count evertime function called
end
```

Ode15sstiff.m

```
% ODE15SSTIFF
%%
% SEE ODEFUN.M FOR FUNCTION "ODEFUN"
%Parameters
tspan=[0,2]; %solve interval
x0=[2;-2;1]; %initial condition
%%
%initialize global parameters
global A
A=[96 2 -99
  101 -3 -99
  196 2 -199];  %set A equal to matrix in lab
global count
count = 0; %reset count to 0

%%
%Solving
opts=odeset('Reltol',2.5e-4);
[t,X]=ode15s(@prime,tspan,x0,opts);

%%
%Plot it
figure(5)
plot (t,X(:,1),'b', t,X(:,2),'r', t,X(:,3),'g','LineWidth',2.0)
xlabel('Time (Units of Time)')
ylabel('Value of X')
title('X(t) Evaluated with ODE15S')
legend('x(t)','y(t)','z(t)') %config to make good plot

%Analytics
nsteps=length(t)
count
X2s=X(end,:)
error=max(max((abs(X2exact-X2s)))) %compute error max
```

Ode45stiff.m

```
% ODE45STIFF
%%
% SEE ODEFUN.M FOR FUNCTION "ODEFUN"
%Parameters
tspan=[0,2];%solve interval
x0=[2;-2;1];%initial condition
%%
%initialize global parameters
global A
A=[96 2 -99
  101 -3 -99
  196 2 -199];  %set A equal to matrix in lab
global count
count = 0; %reset count to 0
%%
%Solving
opts=odeset('Reltol',1.75e-4); %error tol from lab
[t,X]=ode45(@prime,tspan,x0,opts);

%%
%Plot it
figure(4)
plot (t,X(:,1),'b', t,X(:,2),'r', t,X(:,3),'g','LineWidth',2.0)
xlabel('Time (Units of Time)')
ylabel('Value of X')
title('X(t) Evaluated with ODE45')
legend('x(t)','y(t)','z(t)') %config to make good plot

%Analytics
nsteps=length(t)
count
X2s=X(end,:)
error=max(max((abs(X2exact-X2s)) %compute error max
```

Appendix 2: Code for Tasks 2.5

prime2.m

```
%prime2.m
function dxdt = prime2(t,X)
global count k1 k2 k3 k4 A B %initialize globals
dxdt=[k1*A-k2*B*X(1)+k3*X(1)^2*X(2)-k4*X(1);k2*B*X(1)-k3*X(1)^2*X(2)];
count=count+1;
end
```

ode15sbruss.m

```
%ode15sbruss
%%
% SEE PRIME2.M FOR FUNCTION "ODEFUN"
%Parameters
tspan=[0,400];     %solve interval
x0=[0,0];    %initial condition
%%
%initialize global parameters

global count k1 k2 k3 k4 A B

k1 = []; k1vec=[1.5,1.4,1.2,0.7,0.4,0.2];
k2 = 1; %paramaterize these as given in the lab
k3 = 1;
k4 = 1;
A = 1;
B = 3;

for ii=1:length(k1vec)
      %%
      count = 0; %reset count to 0
      k1 = k1vec(ii); %use the first k1 parameter in list

      %Solving
      opts=odeset('Reltol',10e-8); %set solver options
    [t,X]=ode15s(@prime2,tspan,x0,opts); %call solver

      %%
      %Plot solution curves
      figure(5+ii)
      clf
      plot (t,X(:,1),'b', t,X(:,2),'--r','LineWidth',2.0)
      xlabel('Time (Units of Time)')
      ylabel('Value of Solution')
      title(append('X(t) and Y(t) Evaluated with ODE15S for k1 =
',num2str(k1vec(ii))));
      legend('X(t)','Y(t)') %config to make good plot
      if ii==1 %set limits of plot based on lab report
      xlim([0,400])
      ylim([0,4])
      elseif ii==2||3||4
      xlim([200,400])
      ylim([0,8])
      elseif ii==5||6
      xlim([0,400])
      ylim([0,25])
```

```
      end
      %%
      %Plot phase diagrams
      figure(12+ii)
      clf
      plot(X(:,1),X(:,2),'k','LineWidth',2.0)
      xlabel('X(t)')
      ylabel('Y(t)')
      title(append('Phase Diagram for X(t) and Y(t) Evaluated with ODE15S for
k1 = ',num2str(k1vec(ii))));
      xlim([0,25])
      ylim([0,25])
      if ii<4
      xlim([0,5])
      ylim([0,5])
      end
      hold on %create red circle for critical point
    plot((k1*A)/k4,(k2*B*k4)/(k3*k1*(A)),'-mo','MarkerEdgeColor','r',...
%define corrdinates of critical point
      'MarkerFaceColor','r','MarkerSize',5);
      legend(append('k1= ',num2str(k1vec(ii))),'Critical Point'); %insert a
legend
      hold off
      %%
      %Analytics
      nsteps(ii)=length(t); %compute tsteps and count and store them in an
array
      countvec(ii)=count;
end %repeat
```

ode45bruss.m

```
%ode45bruss
%%
% SEE PRIME2.M FOR FUNCTION "ODEFUN"
%Parameters
tspan=[0,400];     %solve interval
x0=[0,0];    %initial condition
%%
%initialize global parameters

global count k1 k2 k3 k4 A B

k1 = []; k1vec=[1.5,1.4,1.2,0.7,0.4,0.2];
k2 = 1; %paramaterize these as given in the lab
k3 = 1;
k4 = 1;
A = 1;
B = 3;

for ii=1:length(k1vec)
      %%
      count = 0; %reset count to 0
      k1 = k1vec(ii); %use the first k1 parameter in list

      %Solving
      opts=odeset('Reltol',10e-8); %set solver options
    [t,X]=ode45(@prime2,tspan,x0,opts); %call solver

      %%
      %Plot solution curves
      figure(5+ii)
      plot (t,X(:,1),'b', t,X(:,2),'--r','LineWidth',2.0)
      xlabel('Time (Units of Time)')
      ylabel('Value of X')
      title(append('X(t) and Y(t) Evaluated with ODE45 for k1 =
',num2str(k1vec(ii))));
      legend('X(t)','Y(t)') %config to make good plot
      if ii==1 %set limits of plot based on lab report
      xlim([0,400])
      ylim([0,4])
      elseif ii==2||3||4
      xlim([200,400])
      ylim([0,8])
      elseif ii==5||6
      xlim([0,400])
      ylim([0,25])
      end
```

```
      %%
      figure(12+ii)
      clf
      plot(X(:,1),X(:,2),'k','LineWidth',2.0)
      xlabel('X(t)')
      ylabel('Y(t)')
      title(append('Phase Diagram for X(t) and Y(t) Evaluated with ODE15S for
k1 = ',num2str(k1vec(ii))));
      xlim([0,25])
      ylim([0,25])
      if ii<4
      xlim([0,5])
      ylim([0,5])
      end
      hold on %create red circle for critical point
    plot((k1*A)/k4,(k2*B*k4)/(k3*k1*(A)),'-mo','MarkerEdgeColor','r',...
%define corrdinates of critical point
      'MarkerFaceColor','r','MarkerSize',5);
      legend(append('k1= ',num2str(k1vec(ii))),'Critical Point'); %insert a
legend
      hold off
      %%
      %Analytics
      nsteps(ii)=length(t); %compute tsteps and count and store them in an
array
      countvec(ii)=count;
end %repeat
```

Appendix 3: Code for Tasks 2.6

prime2.m

```
%prime2.m
function dxdt = prime2(t,X)
global count k1 k2 k3 k4 A B %initialize globals
dxdt=[k1*A-k2*B*X(1)+k3*X(1)^2*X(2)-k4*X(1);k2*B*X(1)-k3*X(1)^2*X(2)];
count=count+1;
end
```

ode15smodified.m

```
%task 7.m
%ode15sbrussmodified
%%
% SEE PRIME2.M FOR FUNCTION "ODEFUN"
%Parameters
tspan=[0,400];     %solve interval
x0=[0,0;1,2;2,1];  %initial condition
%%
%initialize global parameters

global count k1 k2 k3 k4 A B

k1 = 1.2;
k2 = 1;
k3 = 1;
k4 = 1;
A = 1;
B = 3;

      %%
      count = 0; %reset count to 0
      %Solving
      opts=odeset('Reltol',10e-8);
    [t1,X1]=ode15s(@prime2,tspan,x0(1,:),opts);
    [t2,X2]=ode15s(@prime2,tspan,x0(2,:),opts);
    [t3,X3]=ode15s(@prime2,tspan,x0(3,:),opts);
      %%
      %Plot solution curves for X(t)
      figure(1)
      clf
      plot (t1,X1(:,1),'b',t2,X2(:,1),'--g',t3,X3(:,1),'.r','LineWidth',2.0)
      xlabel('Time (Units of Time)')
      ylabel('Value of X')
      title('X(t) Evaluated with ODE15S for different ICs');
      legend('X0=0,Y0=0','X0=1,Y0=2','X0=2,Y0=1') %config to make good plot
      xlim([350,400])
      ylim([0,4])


      %%
      %Plot solution curves for X(t)
      figure(2)
      clf
      plot (t1,X1(:,2),'b',t2,X2(:,2),'--g',t3,X3(:,2),'.r','LineWidth',2.0)
      xlabel('Time (Units of Time)')
      ylabel('Value of Y')
```

```
        title('Y(t) Evaluated with ODE15S for different ICs');
        legend('X0=0,Y0=0','X0=1,Y0=2','X0=2,Y0=1') %config to make good plot
        xlim([350,400])
        ylim([0,4])


        %Plot the phase diagram
        figure(3)
        clf

plot(X1(:,1),X1(:,2),'b',X2(:,1),X2(:,2),'--g',X3(:,1),X3(:,2),'.r','LineWidth'
,2.0)
        xlabel('X(t)')
        ylabel('Y(t)')
        title('Phase Diagram for X(t) and Y(t) Evaluated with ODE15S for
different ICs');
        xlim([0,5])
        ylim([0,5])
```