MATH0086 Exercise 1

Nikolay Walters

The Brusselator

1 Introduction

This report is a summary of my attempts to solve the Brusselator model using numerical methods and to investigate its properties. The Brusselator model is described by the following differential equations:

$$\frac{dx}{dt} = A - Bx + x^2y - x \tag{1}$$

$$\frac{dy}{dt} = Bx - x^2y \tag{2}$$

with positive constants A and B. The Brusselator has a fixed point at $x_{eq} = A$ and $y_{eq} = \frac{B}{A}$, which becomes unstable if $B > 1 + A^2$ resulting in an oscillatory behaviour. Runge-Kutta methods are suitable for both stable decay to equilibrium (i.e. heat equation) and time-symmetric (i.e. harmonic oscillator) regimes making it applicable to the Brusselator model.

2 Part A

2.1 Fourth-Order Accurate Runge-Kutta

For a general problem given by $\dot{y} = f(t, y), \quad y(t_0) = y_0$, where y is the unknown function of t, $\dot{y}$ is a time derivative of y and t_0 and y_0 are the initial conditions, the classical fourth-order Runge-Kutta (RK4) method takes the following steps:

- $k_1 = hf(t_n, y_n)$
- $k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})$
- $k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2})$
- $k_4 = hf(t_n + h, y_n + k_3)$
- $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
- $t_{n+1} = t_n + h$

In the above equations, $t_n = nh$ with a time step h for $n = 1, 2, 3 \ldots$ on a uniform grid. The RK4 approximation of $y(t_{n+1})$ is y_{n+1}. Such methodology is widely used, and can be found in Hairer et al. (1989)[1]. As can be seen, it is an explicit method.

To extend this algorithm to a system of two equations, thus making it applicable to the Brusselator we must take great care with the dependencies. One way to generalize such an algorithm for a system of equations can be achieved by

re-writing it in the vector form. Let us define RHS of (1) as $f(t, x, y)$ and RHS of (2) as $g(t, x, y)$. The $\bar{k}_n$ increments now become a vector with, in our case, two components (k_n and l_n). Based on our model, $f(t, x, y)$ will be associated with k increments and $g(t, x, y)$ with l. This results in the following algorithm:

- $k_1 = hf(t_n, x_n, y_n)$
- $l_1 = hg(t_n, x_n, y_n)$
- $k_2 = hf(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}l_1)$
- $l_2 = hg(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_1, y_n + \frac{1}{2}l_1)$
- $k_3 = hf(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}l_2)$
- $l_3 = hg(t_n + \frac{1}{2}h, x_n + \frac{1}{2}k_2, y_n + \frac{1}{2}l_2)$
- $k_4 = hf(t_n + h, x_n + k_3, y_n + l_3)$
- $l_4 = hg(t_n + h, x_n + k_3, y_n + l_3)$
- $x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
- $y_{n+1} = y_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4)$

2.2 Matlab Code

Code shown below uses the methodology outlined in the previous section to find an approximation for the Brusselator model. It also plots $f(t, x, y)$ and $g(t, x, y,)$ for a specified time range, t. The initial conditions used are: $x_0 = 0$ and $y_0 = 1$. Constant A is 2 and constant B is 6. Time step h is equal to 0.01 and t range is from 0 to 25.

```
clear all
tmax = 25;  % t range
step = 0.01; % h value
tsol = 0:step:tmax; % t array
ysol = zeros(1, length(tsol)); % y array
xsol = zeros(1, length(tsol)); % x array
ysol(1) = 1; % initial y
xsol(1) = 0; % initial x
Aconst = 2; % A constant
Bconst = 6; % B constant
fxyt = @(tsol, xsol, ysol) Aconst - (Bconst * xsol) + xsol^2 * ysol - xsol; % f(t,x,y)
gxyt = @(tsol, xsol, ysol) Bconst * xsol - xsol^2 * ysol; % g(t,x,y)
for i = 1:(length(tsol)-1) % RK4
    k0 = fxyt(tsol(i), xsol(i), ysol(i));
    l0 = gxyt(tsol(i), xsol(i), ysol(i));
    k1 = fxyt(tsol(i)+0.5*step,xsol(i)+0.5*step*k0,ysol(i)+0.5*step*l0);
    l1 = gxyt(tsol(i)+0.5*step,xsol(i)+0.5*step*k0,ysol(i)+0.5*step*l0);
    k2 = fxyt((tsol(i)+0.5*step),(xsol(i)+0.5*step*k1),(ysol(i)+0.5*step*l1));
    l2 = gxyt((tsol(i)+0.5*step),(xsol(i)+0.5*step*k1),(ysol(i)+0.5*step*l1));
    k3 = fxyt((tsol(i)+step),(xsol(i)+k2*step),(ysol(i)+l2*step));
    l3 = gxyt((tsol(i)+step),(xsol(i)+k2*step),(ysol(i)+l2*step));
    xsol(i+1) = xsol(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3); % x_{n+1}
```

```
    ysol(i+1) = ysol(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3); % y_{n+1}
end
hold on
title('RK4 approximation of the Brusselator between t = 0 and 25');
xlabel('t');
ylabel('Approximation');
plot(tsol, xsol,'DisplayName', 'x');
plot(tsol, ysol, 'DisplayName', 'y');
legend('show')
hold off
```

The output of this code is a single graph showing $x(t)$ and $y(t)$:

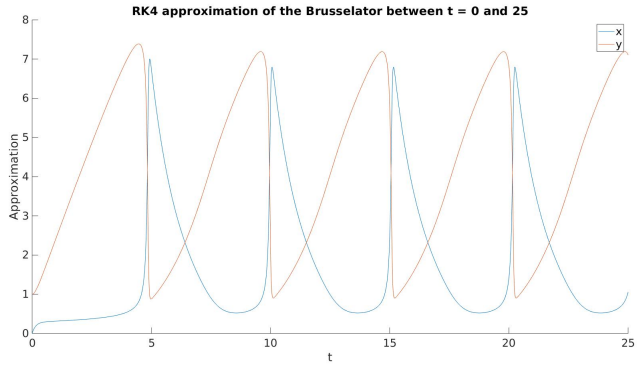

Fig. 1: $x(t)$ and $y(t)$ with $A = 2$, $B = 6$, $h = 0.01$ and $0 \leq t \leq 25$.

Plotting $x(t)$ against $y(t)$ for the same t range and parameters as before results in the following plot:

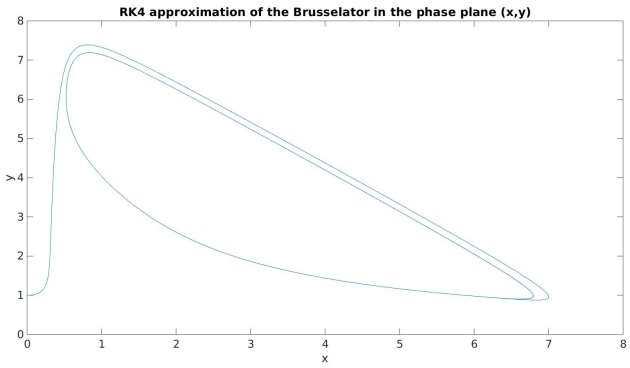

Fig. 2: $x(t)$ against $y(t)$ with $A = 2$, $B = 6$, $h = 0.01$ and $0 \leq t \leq 25$.

The t range employed in these diagrams is deemed acceptable for the following reasons. Firstly, Fig. 1 shows at least 3 full periods for the functions. Thus, it can be confirmed that they do indeed show repeating pattern with a time period (p) of approximately $5t$ units. Secondly, in Fig. 2 a clear convergence can be seen very early on. A limit cycle is approached to a good accuracy at around $t = 11$ or 2 periods.

At step of $h = 0.1$ local instability can be seen in both $x(t)$ and $y(t)$ at approximately $t = 5 + np$ for $n = 0, 1, 2, \ldots$ where the rate of change of $x(t)$ and $y(t)$ appears to be the greatest. These can be seen in the figure below (Fig. 3).

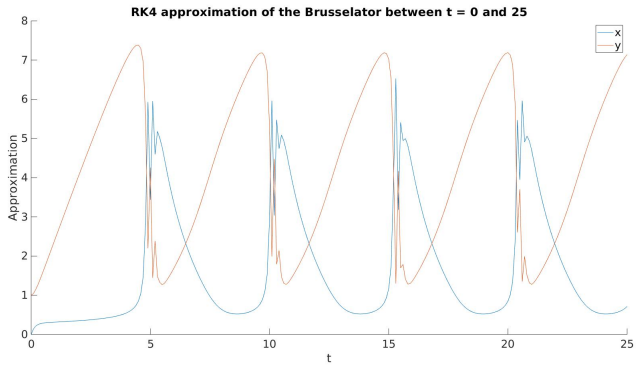

Fig. 3: $x(t)$ and $y(t)$ with $A = 2$, $B = 6$, $h = 0.1$, $0 \leq t \leq 25$ and local ODE dependant instability.

At step size of $h = 0.02$ the local instability (very small oscillations around the actual solution) becomes impossible to see on the $x(t)$ and $y(t)$ against t plot but can still be barely distinguished in the phase space diagram. Fig. 4 shows

both diagrams. Such effects become unnoticeable to the bare eye at $h = 0.01$. Therefore, $h = 0.01$ is deemed an acceptable choice, since no local instabilities can be seen while the step size remains relatively large for computational efficiency.

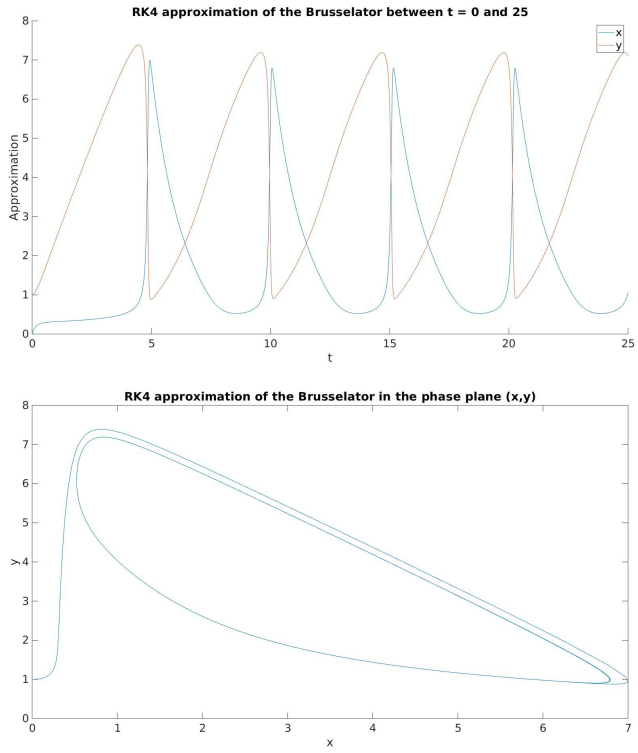

Fig. 4: Top: $x(t)$ and $y(t)$ with $A = 2$, $B = 6$, $h = 0.02$, $0 \leq t \leq 25$ and an unnoticeable instability. Bottom: $x(t)$ against $y(t)$ with $A = 2$, $B = 6$, $h = 0.02$, $0 \leq t \leq 25$ and a barely noticeable instability in the bottom right quadrant. Such artifact is expressed as a slightly thicker line around that point due to indistinguishable oscillations being stacked on top of each other.

These step sizes are of course only applicable to the range of the parameters specified earlier. We could render local instability with $h = 0.01$ by, for example, setting the value of B to 9 while keeping the other parameters as before, as shown in Fig. 5. Finally, larger step sizes, such as $h = 0.25$ are completely unstable. Such approximations reach maximum numerical values that can be stored by a variable extremely quickly and therefore provide little practical application.

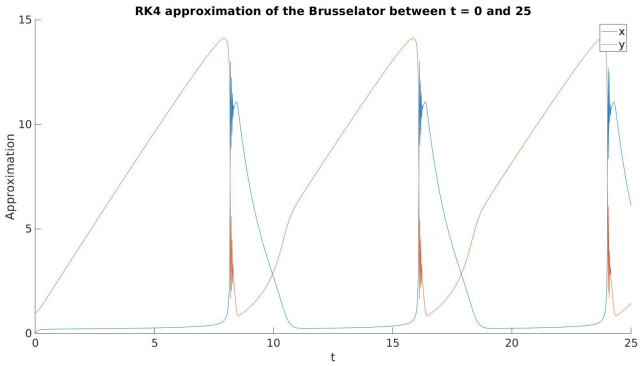

Fig. 5: $x(t)$ and $y(t)$ with $A = 2$, $B = 9$, $h = 0.01$, $0 \leq t \leq 25$ and an local instability.

2.3 Limit Cycle as a Global Attractor

Taking a number of initial x_0 and y_0 conditions in the first quadrant demonstrates that the limit cycle is a global attractor as shown in Fig. 6. Also note that taking y_0, x_0 inside the limit cycle also results in the convergence on the limit cycle. This results in interesting x/y vs. t diagrams where the initial small oscillation amplifies itself before reaching a constant amplitude. There are also cases when the initial conditions are so far off the limit cycle that the solution diverges rapidly. An example of the phase space where the solution just manages to converge on the limit cycle is shown in Fig. 7. While the oscillations are caused by the initial values lying close to the stability boundary, it is reassuring to see that the limit cycle still acts as a global attractor after a good 'shake'.

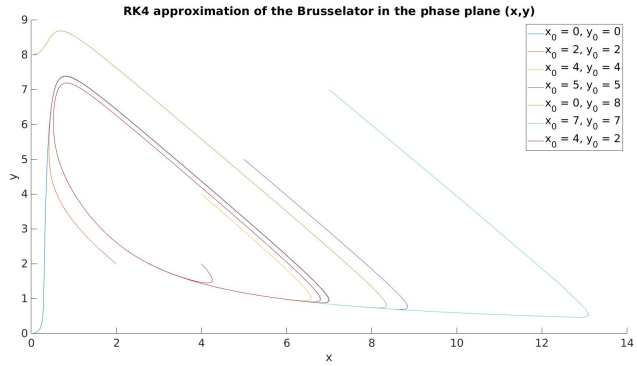

Fig. 6: A number of initial conditions resulting in a convergence at the limit cycle with $A = 2$, $B = 6$, $h = 0.01$ and $0 \leq t \leq 25$.

The only initial condition for which the limit cycle does not act as a global attractor is the equilibrium point, i.e. $x_0 = x_{eq} = A$ and $y_0 = y_{eq} = \frac{B}{A}$. Unsurprisingly, this results in $y_{n+1} = y_n = y_{eq}$ and $x_{n+1} = x_n = x_{eq}$ for $n \in \mathbb{N}$. Taking initial values just outside of the equilibrium results in a divergence from it into the limit cycle.

2.4 Stability

It is possible to estimate a stability 'boundary' using numerical methods for a particular set of conditions. The easiest (if not the only) way is to compute a discrete mesh that checks for convergence at the vertices of every segment. I only investigated the first quadrant and I do not expect the boundary to be symmetric. Fig. 8 shows a rough estimated boundary, beyond which the solution does not converge. The boundary is clearly non-linear. In fact, it seems likely that no simple function can describe the boundary well. Moreover, the boundary itself is not a clear-cut separation. I noticed multiple 'pockets' of stability outside of

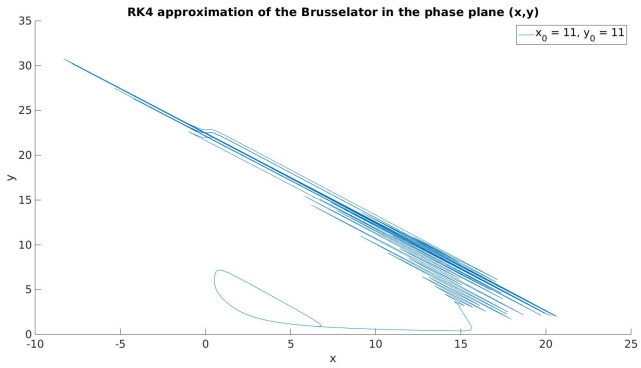

Fig. 7: x_0 and y_0 both set to 11 result in oscillatory behaviour before converging on the limit cycle with $A = 2$, $B = 6$, $h = 0.01$ and $0 \leq t \leq 25$.

the boundary and 'pockets' of instability close to but inside the boundary. The boundary itself appears to be highly granulated. This can be problematic in certain cases, so I estimated a strict boundary described by a straight line that excludes all of the granular behaviour. In other words, between the origin and the strict boundary we do not expect to see any local instability pockets.

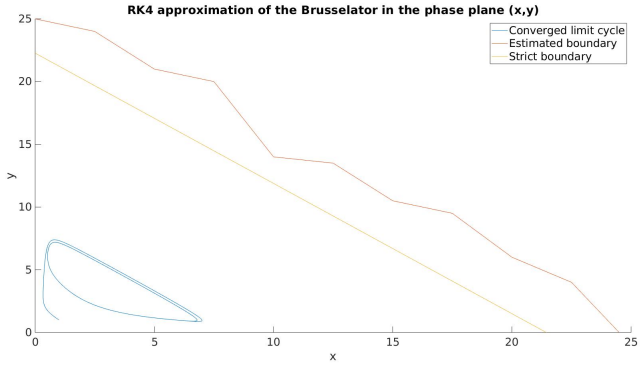

Fig. 8: Stability region for $A = 2$, $B = 6$, $h = 0.01$ and $0 \leq t \leq 25$. Red line shows an estimated stability boundary. Orange line shows the strict stability boundary that contains no instability pockets. Location of the limit cycle is shown by the blue line.

An ODE is called stable if its Jacobian is a negative real value at equilibrium. To extend this theorem to a system, we need to calculate the Jacobian matrix and find its eigenvalues. For a system of two ODEs the Jacobian (J)

is $\begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix}$. Substituting $A - Bx + x^2y - x$ as f_1 and $Bx - x^2y$ as f_2 and taking partial derivatives results in $J = \begin{pmatrix} -B + 2xy - 1 & x^2 \\ B - 2xy & -x^2 \end{pmatrix}$. Evaluating at equilibrium points ($x_{eq} = A$ and $y_{eq} = \frac{B}{A}$) results in $\begin{pmatrix} B - 1 & A^2 \\ -B & -A^2 \end{pmatrix}$. The resulting general eigenvalue equation is $\lambda_{\pm} = \frac{1}{2}((B - A - 1) \pm \sqrt{(B - A - 1)^2 - 4A})$. Substituting $A = 2$ and $B = 6$ results in both eigenvalues λ being real and positive. Therefore, for this set of parameters, the system of ODEs is unstable. On the other hand, $B - A < 1$ would result in both the eigenvalues having negative real parts, implying the system would be stable.

Next, we can determine the stability region for RK4. To do this, we need to determine the stability function (g) and identify the region with $|g| < 1$ by applying RK4 to the linear test equation $y' = \lambda y$. RK4 gives $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$. Applying on the test equation we obtain $k_1 = h\lambda y_n$, $k_2 = h\lambda(y_n + \frac{h\lambda y_n}{2})$, $k_3 = h\lambda(y_n + \frac{y_n + \frac{h\lambda y_n}{2}}{2})$, etc. Substituting k_n into y_{n+1} equation and factorising y_n out results in $y_{n+1} = (1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4)y_n = g(h\lambda)y_n$. Let $h\lambda = z = x + iy$. We can now numerically solve $|1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4| < 1$ using a mesh for multiple x and y values. Fig. 9 shows the stability boundary for RK4 in the complex plane, within which $|g(z)| < 1$. It is clear that the stability function is step (h) dependant and so we expect RK4 to be conditionally stable. Theoretically, we could find a stability constraint for h if we could express the Brusselator model as $\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}$, but practically, this seems to be impossible for me. Either way, the stability boundary estimated above and the previous numerical estimates in the $x - y$ plane of the Brusselator model act as solid evidence of conditional stability. In other words, we require $B - A < 1$ and reasonably small h to achieve stability. Otherwise, unless initial conditions are well picked the solution diverges.

2.5 Accuracy of the Solution

We know that the RK4 is 4th-order accurate, thus we expect the global truncation error of $O(h^4)$. Therefore, we should expect to see roughly 10^4 improvement in the accuracy if we reduce step size h by 10. We could test it by taking a range of h values and seeing if the above statement is realistic. I used $h = 0.0001, 0.001, 0.01, 0.1$ and 1 with $A = 1.5$ and $B = 2$ so that we work in the stable region. I then averaged out the difference between $h = 0.0001 = h_c$ and a larger step with $t_{max} = 25$, $x_0 = 1$ and $y_0 = 1$. Then by taking the log to the base of 10 of this averaged difference and plotting it we should expect to see a approximately straight line with gradient of 4. Fig. 10 displays the result. The actual gradient of the line using least-squares method is 3.897, which deemed a satisfactory proof of $O(h^4)$ behaviour. Code for this is given in Appendix 1.

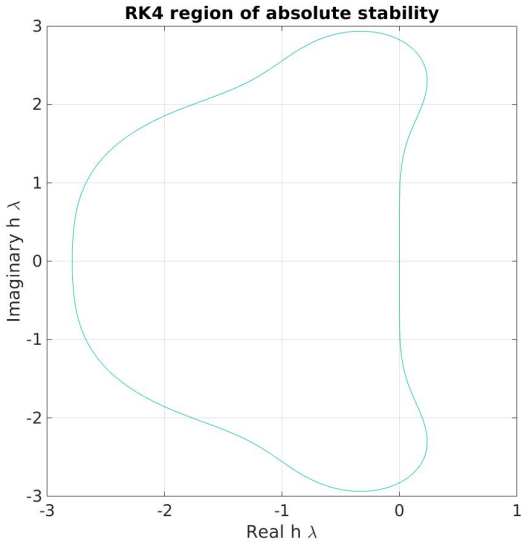

Fig. 9: Absolute stability boundary for RK4.

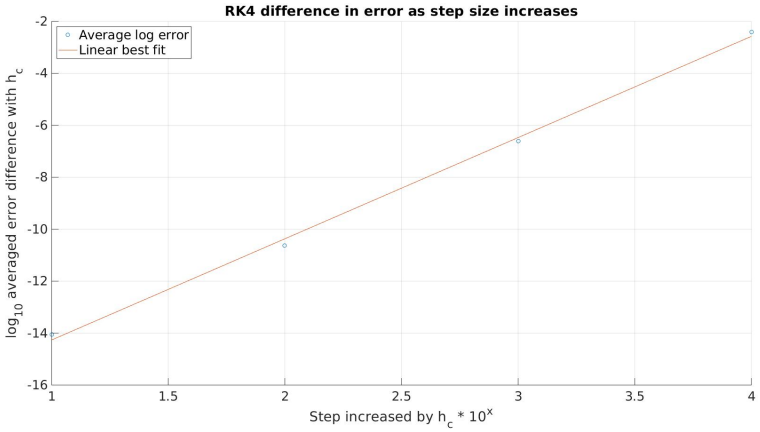

Fig. 10: Averaged $\log_{10}$ difference between $h_c = 0.0001$ and larger h steps (blue circles). Linear best fit shown in red.

The averaged errors between each $\times 10 h_c$ increment and h_c are 8.856×10^{-15}, 2.393×10^{-11}, 2.478×10^{-7} and 4.000×10^{-3}. Their corresponding $\log_{10}$ values are -14.05, -10.62, -6.61 and -2.40 respectively.

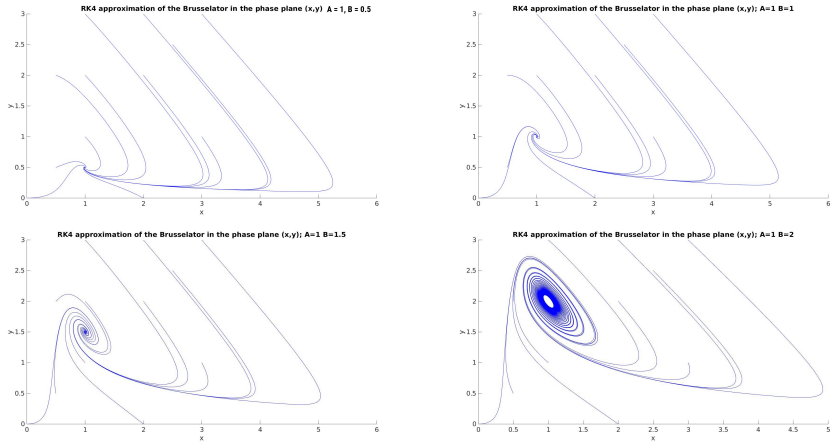

Fig. 11: $x(t)$ and $y(t)$ approximations to the Brusselator for a small range of $B \leqslant 1+A^2$ conditions.

3 Part B

One obvious choice of A and B parameters that have not been investigated in detail so far is the stable regime condition with $B < 1 + A^2$. $B = 1 + A^2$ is also of interest. Fig. 11 shows $x(t)$ against $y(t)$ for $A = 1.0$ and $B = 0.5, 1.0, 1, 5$ and 2.0. Initial x_0 and y_0 used are shown in Table 1. h was kept constant and set to 0.01, with t range between 0 and 250. Fig. 12 then shows $x(t)$ and $y(t)$ as a function of t for the same A and B selection with $A = 1$, $B = 2.5$ (unstable regime) added for a comparison. For Fig. 12 only $x_0 = 1$ and $y_0 = 1$ were plotted to not overcloud the figure.

As can be seen in Fig. 11 point $x = A$, $y = B/A$ acts as a global attractor equilibrium point, with the solution converging on it. However, such behaviour cannot be seen at $A = 1$, $B = 2$, with the function approaching the equilibrium slower and slower as t increases. It seems that the smaller the B is compared to $1 + A^2$, the quicker the solution converges on that point, with $B = 1 + A^2$ converging at $t = \infty$. Overall, the system can be described as stable as long as inequality holds.

Fig. 12 shows that the amplitude of oscillations slowly decreases as long as the stability inequality holds. Unstable regime with $A = 1$ and $B = 2.5$ clearly shows no visible amplitude decrease, as expected from the limit cycle behaviour.

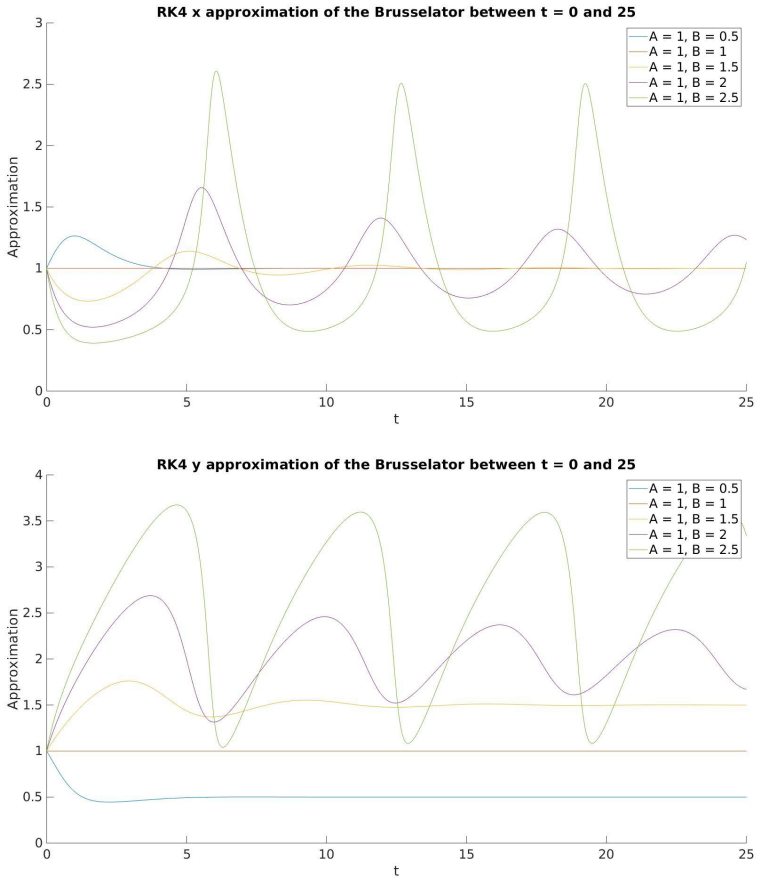

Fig. 12: Top: $x(t)$ against t for a selection of A, B conditions. Bottom: the same but for $y(t)$.

x_0	y_0
0	0
0.5	0.5
0.5	2
1	1
1	2
1	3
2	0
2	2
2	3
2.5	2.5
3	1
3	3

Table 1: Initial x_0 and y_0 used to create Fig. 11.

On the other hand, the unstable $A = 1$, $B = 2$ does show a decrease in amplitude, which is likely to converge at $t = \infty$.

Overall, this short work provides an overview on the numerical solutions of the Brusselator model using the RK4 method. A number of initial conditions were explored, covering points of interests and different regimes. The basic outline of the Brusselator and its modes was also given.

References

[1] Hairer, E., Christian Lubich, and Michel Roche. The Numerical Solution of Differential-algebraic Systems by Runge-Kutta Methods / Ernst Hairer, Christian Lubich, Michel Roche. Berlin ; New York: Springer-Verlag, 1989. Print. Lecture Notes in Mathematics (Springer-Verlag) ; 1409.

A Best fit line code

```
tmax = 25;
% a loop of h values would do much better here
% method below is equivalent to the code given in the report
step = 0.0001;
tsol = 0:step:tmax;
ysol = zeros(1, length(tsol));
xsol = zeros(1, length(tsol));
xsol(1) = 1;
ysol(1) = 1;
Aconst = 1.5;
Bconst = 2;
fxyt = @(tsol, xsol, ysol) Aconst - (Bconst * xsol) + xsol^2 * ysol - xsol;
gxyt = @(tsol, xsol, ysol) Bconst * xsol - xsol^2 * ysol;
for i = 1:(length(tsol)-1)
    k0 = fxyt(tsol(i), xsol(i), ysol(i));
    l0 = gxyt(tsol(i), xsol(i), ysol(i));
    k1 = fxyt(tsol(i)+0.5*step,xsol(i)+0.5*step*k0,ysol(i)+0.5*step*l0);
    l1 = gxyt(tsol(i)+0.5*step,xsol(i)+0.5*step*k0,ysol(i)+0.5*step*l0);
    k2 = fxyt((tsol(i)+0.5*step),(xsol(i)+0.5*step*k1),(ysol(i)+0.5*step*l1));
    l2 = gxyt((tsol(i)+0.5*step),(xsol(i)+0.5*step*k1),(ysol(i)+0.5*step*l1));
    k3 = fxyt((tsol(i)+step),(xsol(i)+k2*step),(ysol(i)+l2*step));
    l3 = gxyt((tsol(i)+step),(xsol(i)+k2*step),(ysol(i)+l2*step));
    xsol(i+1) = xsol(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3);
    ysol(i+1) = ysol(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3);
end
step = 0.001;
tsol0 = 0:step:tmax;
ysol0 = zeros(1, length(tsol0));
xsol0 = zeros(1, length(tsol0));
xsol0(1) = 1;
ysol0(1) = 1;
Aconst = 1.5;
Bconst = 2;
fxyt = @(tsol0, xsol0, ysol0) Aconst - (Bconst * xsol0) + xsol0^2 * ysol0 - xsol0;
gxyt = @(tsol0, xsol0, ysol0) Bconst * xsol0 - xsol0^2 * ysol0;
for i = 1:(length(tsol0)-1)
    k0 = fxyt(tsol0(i), xsol0(i), ysol0(i));
    l0 = gxyt(tsol0(i), xsol0(i), ysol0(i));
    k1 = fxyt(tsol0(i)+0.5*step,xsol0(i)+0.5*step*k0,ysol0(i)+0.5*step*l0);
    l1 = gxyt(tsol0(i)+0.5*step,xsol0(i)+0.5*step*k0,ysol0(i)+0.5*step*l0);
    k2 = fxyt((tsol0(i)+0.5*step),(xsol0(i)+0.5*step*k1),(ysol0(i)+0.5*step*l1));
    l2 = gxyt((tsol0(i)+0.5*step),(xsol0(i)+0.5*step*k1),(ysol0(i)+0.5*step*l1));
    k3 = fxyt((tsol0(i)+step),(xsol0(i)+k2*step),(ysol0(i)+l2*step));
```

```
    l3 = gxyt((tsol0(i)+step),(xsol0(i)+k2*step),(ysol0(i)+l2*step));
    xsol0(i+1) = xsol0(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3);
    ysol0(i+1) = ysol0(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3);
end
step = 0.01;
tsol1 = 0:step:tmax;
ysol1 = zeros(1, length(tsol1));
xsol1 = zeros(1, length(tsol1));
xsol1(1) = 1;
ysol1(1) = 1;
Aconst = 1.5;
Bconst = 2;
fxyt = @(tsol1, xsol1, ysol1) Aconst - (Bconst * xsol1) + xsol1^2 * ysol1 - xsol1;
gxyt = @(tsol1, xsol1, ysol1) Bconst * xsol1 - xsol1^2 * ysol1;
for i = 1:(length(tsol1)-1)
    k0 = fxyt(tsol1(i), xsol1(i), ysol1(i));
    l0 = gxyt(tsol1(i), xsol1(i), ysol1(i));
    k1 = fxyt(tsol1(i)+0.5*step,xsol1(i)+0.5*step*k0,ysol1(i)+0.5*step*l0);
    l1 = gxyt(tsol1(i)+0.5*step,xsol1(i)+0.5*step*k0,ysol1(i)+0.5*step*l0);
    k2 = fxyt((tsol1(i)+0.5*step),(xsol1(i)+0.5*step*k1),(ysol1(i)+0.5*step*l1));
    l2 = gxyt((tsol1(i)+0.5*step),(xsol1(i)+0.5*step*k1),(ysol1(i)+0.5*step*l1));
    k3 = fxyt((tsol1(i)+step),(xsol1(i)+k2*step),(ysol1(i)+l2*step));
    l3 = gxyt((tsol1(i)+step),(xsol1(i)+k2*step),(ysol1(i)+l2*step));
    xsol1(i+1) = xsol1(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3);
    ysol1(i+1) = ysol1(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3);
end
step = 0.1;
tsol2 = 0:step:tmax;
ysol2 = zeros(1, length(tsol2));
xsol2 = zeros(1, length(tsol2));
xsol2(1) = 1;
ysol2(1) = 1;
Aconst = 1.5;
Bconst = 2;
fxyt = @(tsol2, xsol2, ysol2) Aconst - (Bconst * xsol2) + xsol2^2 * ysol2 - xsol2;
gxyt = @(tsol2, xsol2, ysol2) Bconst * xsol2 - xsol2^2 * ysol2;
for i = 1:(length(tsol2)-1)
    k0 = fxyt(tsol2(i), xsol2(i), ysol2(i));
    l0 = gxyt(tsol2(i), xsol2(i), ysol2(i));
    k1 = fxyt(tsol2(i)+0.5*step,xsol2(i)+0.5*step*k0,ysol2(i)+0.5*step*l0);
    l1 = gxyt(tsol2(i)+0.5*step,xsol2(i)+0.5*step*k0,ysol2(i)+0.5*step*l0);
    k2 = fxyt((tsol2(i)+0.5*step),(xsol2(i)+0.5*step*k1),(ysol2(i)+0.5*step*l1));
    l2 = gxyt((tsol2(i)+0.5*step),(xsol2(i)+0.5*step*k1),(ysol2(i)+0.5*step*l1));
    k3 = fxyt((tsol2(i)+step),(xsol2(i)+k2*step),(ysol2(i)+l2*step));
    l3 = gxyt((tsol2(i)+step),(xsol2(i)+k2*step),(ysol2(i)+l2*step));
```

```
    xsol2(i+1) = xsol2(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3);
    ysol2(i+1) = ysol2(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3);
end
step = 1;
tsol3 = 0:step:tmax;
ysol3 = zeros(1, length(tsol3));
xsol3 = zeros(1, length(tsol3));
xsol3(1) = 1;
ysol3(1) = 1;
Aconst = 1.5;
Bconst = 2;
fxyt = @(tsol3, xsol3, ysol3) Aconst - (Bconst * xsol3) + xsol3^2 * ysol3 - xsol3;
gxyt = @(tsol3, xsol3, ysol3) Bconst * xsol3 - xsol3^2 * ysol3;
for i = 1:(length(tsol3)-1)
    k0 = fxyt(tsol3(i), xsol3(i), ysol3(i));
    l0 = gxyt(tsol3(i), xsol3(i), ysol3(i));
    k1 = fxyt(tsol3(i)+0.5*step,xsol3(i)+0.5*step*k0,ysol3(i)+0.5*step*l0);
    l1 = gxyt(tsol3(i)+0.5*step,xsol3(i)+0.5*step*k0,ysol3(i)+0.5*step*l0);
    k2 = fxyt((tsol3(i)+0.5*step),(xsol3(i)+0.5*step*k1),(ysol3(i)+0.5*step*l1));
    l2 = gxyt((tsol3(i)+0.5*step),(xsol3(i)+0.5*step*k1),(ysol3(i)+0.5*step*l1));
    k3 = fxyt((tsol3(i)+step),(xsol3(i)+k2*step),(ysol3(i)+l2*step));
    l3 = gxyt((tsol3(i)+step),(xsol3(i)+k2*step),(ysol3(i)+l2*step));
    xsol3(i+1) = xsol3(i) + (1/6)*step*(k0 + 2*k1 + 2*k2 + k3);
    ysol3(i+1) = ysol3(i) + (1/6)*step*(l0 + 2*l1 + 2*l2 + l3);
end

% calculating absolute average log oferror between different steps
errorx = log10(mean(abs(xsol(1:10:end)-xsol0)));
errorx0 = log10(mean(abs(xsol(1:100:end)-xsol1)));
errorx1 = log10(mean(abs(xsol(1:1000:end)-xsol2)));
errorx2 = log10(mean(abs(xsol(1:10000:end)-xsol3)));

% calculating best fit
coefficients = polyfit([1,2,3,4], [errorx, errorx0, errorx1, errorx2],1);
a = coefficients(1); %gradient estimate using least-squares

% plotting error and best fit
hold on;
plot([1;2;3;4], [errorx;errorx0;errorx1;errorx2], 'o', 'DisplayName', 'Average log error');
plot(1:0.1:4, polyval(coefficients,1:0.1:4), '-', 'DisplayName', 'Linear best fit');
title('RK4 difference in error as step size increases','FontSize',14);
xlabel('Step increased by h_c * 10^x', 'FontSize',14);
ylabel('log_{10} averaged error difference with h_c', 'FontSize',14);
grid on;
legend('location', 'northwest', 'FontSize',20);
```

```
set(gca,'FontSize',20);
hold off;
```