# Applied Numerical methods, Group Assignment 1

Klara Zimmerman and Ville Wassberg

September 2023

## Introduction

In this report we use Matlab and freshly acquainted methods to solve different kinds of given problems in the course Applied Numerical methods, SF2520, at KTH. The main focus is to solve the problems numerically with methods presented throughout the course. We begin by using the Runge-Kutta third order method to numerically solve and plot a solution to a magnetisation problem. Later, we consider an ordinary differential equation describing the trajectory of a satellite with the Earth and the Moon as bodies affecting its movement. There we use a multi step method, Adams-Bashforth of order 4. Lastly, we consider the Robertson's problem that is modeling the reactions of three chemicals, and describes the evolution of the concentration of the chemichals over time.

## Problem 1

In this problem we use the Runge-Kutta method to approximate the solution to a Linearized ODE system. The aim is to provide an approximation of the accuracy and stability of the Runge-Kutta method.

We are given the following ODE system:

$$\frac{dm}{dt} = a \times m + \alpha a \times (a \times m),$$

which describes a Landau–Lifshitz equation for the magnetization vector $m(t) \in R^3$, where $\alpha = 0.07$ and $a = \frac{1}{4}[1, \sqrt{11}, 2]^T$. We also get the initial value $m(0) = (0, 0, 1)^T$.

### a)

We begin by computing the components of the vector $m$ with respect to time. This is done by using the numerical method Runge Kutta 3. It is in this lab defined as:

$$k_1 = f(t_n, u_n),$$
$$k_2 = f(t_n + h, u_n + hk_1),$$
$$k_3 = f(t_n + h/2, u_n + hk_1/4 + hk_2/4),$$
$$u_{n+1} = u_n + h/6(k_1 + k_2 + 4k_3), t_n = nh, n = 0, 1, 2, ...$$

The derivative $\frac{dm}{dt}$ is here our function $f$, which in our case is a function of only one variable, $m$. In every step $n$, the approximation of the solution $u_{n+1}$ is calculated using the value $m_n$ in the argument of the function $f$. We implement this method as a MATLAB code, which takes the arguments $m_0$ (initial value), $h$ (time step) and $T$ (end time). The solution is plotted below, for $h = 0.1$ and $T = 50$.

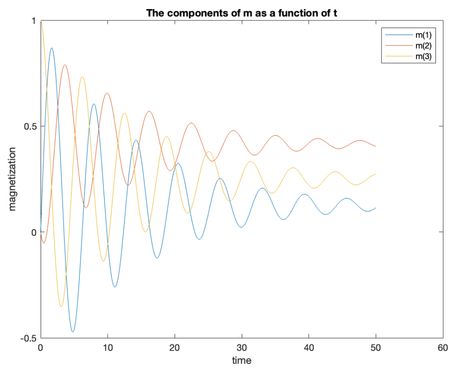

Figure 1: The three components of the solution $m$ as a function of time

Here we plot the trajectory of $m$ with the plot3 command in Matlab. We also plot the given vector $a$ for comparison. As we can see in figure 2 the points of $m(t)$ stays in a plane perpendicular to $a$.

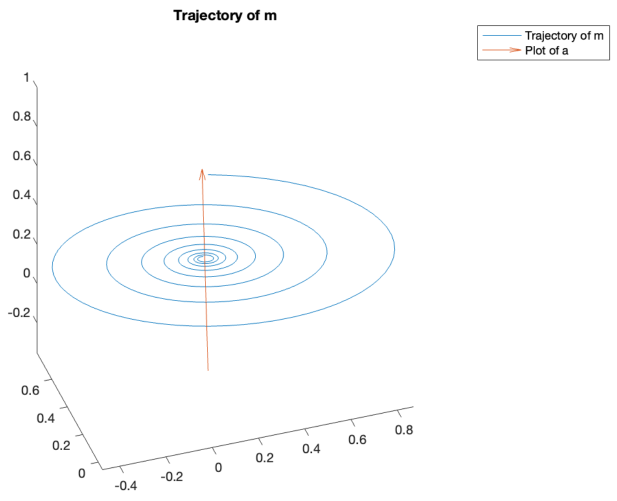

Figure 2: The trajectory of the components of $m$, and the vector $a$

## b)

In this part we study the error of Runge-Kutta 3 as a function of time. We do this by computing the 2-norm difference $|\tilde{m}_N(t) - \tilde{m}_{2N}(t)|_2 = \sqrt{\sum_{k=1}^{3}(\tilde{m}_N - \tilde{m}_{2N})_k^2}$, for $k$ being the $k:th$ component of the vector. In figure 3 we estimate the order of accuracy by comparing the error function to a line with a slope of 3. We note that for small values of $h$, these are parallell, suggesting that the error of the Runge-Kutta method is of order 3 here.

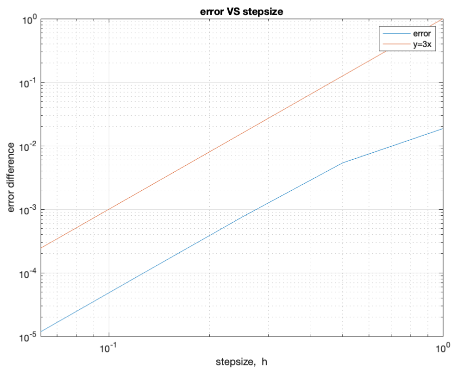

Figure 3: The approximation $|m_N(T) - m_{2N}(T)|$ as a function of $h$.

## c)

To compute the theoretical step size stability limit $h_0$ for the problem, we find the matrix $A \in R^{3\times 3}$ which lets us rewrite our problem as $\frac{dm}{dt} = Am$. By using the formula of the differential above we get the matrix A defined as follows:

$$A = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} + \alpha * \begin{pmatrix} -(a_2^2 + a_3^2) & a_1 a_2 & a_1 a_3 \\ a_1 a_2 & -(a_3^2 + a_1^2) & a_3 a_2 \\ a_1 a_3 & a_2 a_3 & -(a_1^2 + a_2^2) \end{pmatrix}. \quad (1)$$

We are given the stability region, defined as all $z$ in the left half of the complex plane which satisfy the function $s(z) = |1 + z + z^2/2 + z^3/6| - 1 \leq 0$. For every eigenvalue $\lambda_i$ and the stepsize $h$ of the matrix $A$, $s(\lambda_i * h) < 0$ must be satisfied. We created a while-loop which returned the smallest $h$ for which this condition is not satisfied, by starting with a small timestep and increasing it by 0.01 unil an unstable value is found.

For the eigenvalues of $A$, the smallest unstable value we found was $h = 2.07$. In figure 4 the solution using this time step is plotted, along with the stable solution using time step $h = 2.00$.

## d)

In order to varify that the step size stability limit is indeed practically stable, we plot one solution using a slightly smaller step size, and one slightly bigger.

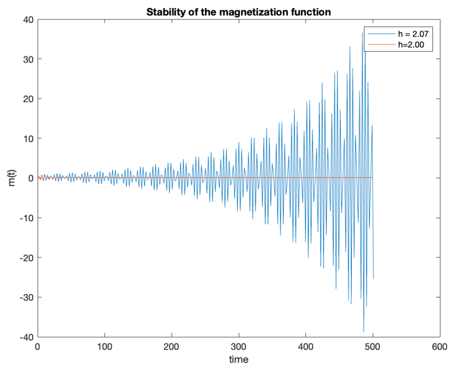

Figure 4: A stable numerical solution of $m(t)$ with timestep $h = 2.00$, and an unstable numerical solution of $m(t)$ with timestep $h = 2.07$, for $t \in [0, 500]$.

## Problem 2

In this problem are we considering a satellite that orbits around the Earth and the moon. We regard the moon and the Earth's centre of mass as the origin in the plot, where the ratio of the masses $\frac{m_{Moon}}{m_{Earth}} = \mu = 1/82.45$, thus, we have Earths centre $\mathbf{c}_0 = (-\mu, 0)$ and the moons $\mathbf{c}_1 = (\mu, 0)$. Hence, the x-axis is the line through $\mathbf{c}_0$ and $\mathbf{c}_1$. The trajectory of the the satellite is ought to be modeled by the following second order ODE:

$$\frac{d^2\mathbf{r}}{dt^2} = -(1-\mu)\frac{\mathbf{r}-\mathbf{c}_0}{|\mathbf{r}-\mathbf{c}_0|^3} - \mu\frac{\mathbf{r}-\mathbf{c}_1}{|\mathbf{r}-\mathbf{c}_1|^3} + 2B + \frac{d\mathbf{r}}{dt} + \mathbf{r}, \tag{2}$$

where $B = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$. The first two terms model the gravitational forces and the last two terms are the centrifugal and the coriolis effect. We are assuming the initial position of the satellite is $\mathbf{r}_0 = (1.15, 0)^T$ with initial velocity $\mathbf{r}_0' = (0, -0.975)^T$.

### a)

In this part of the problem we compute a numerical solution to the equation 2 to track the trajectory of the satellite. In particular, we use the Adams-Bashforth fourth order method:

$$u_{n+1} = \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}, \quad f_n = f(t_n, u_n). \tag{3}$$

Since $f_n$ are dependent on $u_n$ we need three initial $u_i$. We use the Runge-Kutta 3 method from problem 1 to find the next three $\mathbf{u}_i = (\mathbf{r}_i, \frac{d\mathbf{r}_i}{dt_i})^T$ which will be used when implementing the Adams-Bashforth method. Here we begin by plotting the x- and- y-coordinates of the satellites trajectory in the x-y-plane using Matlabs built-in plot method. We also added the Earth's and the moon's centres of mass. The plots are made with a time step $h = 1/2000$ and time cycle $T_{end} = 10$.

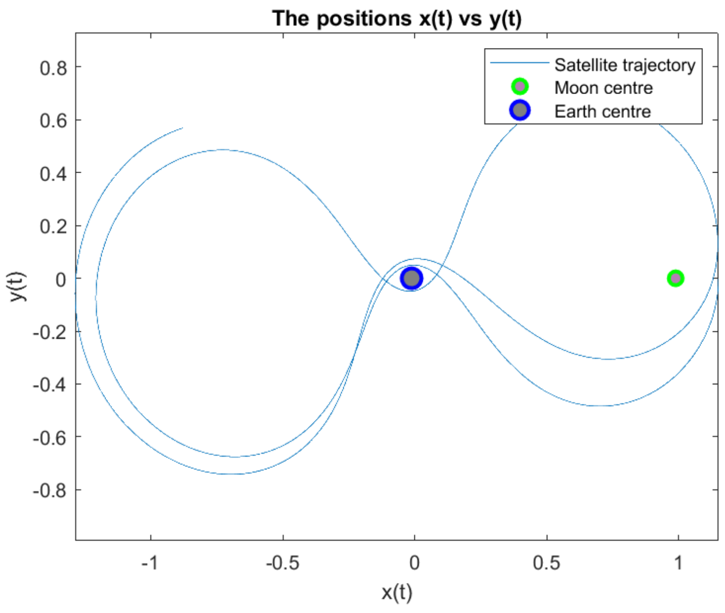

Figure 5: The satellite trajectory where the x and y coordinates are plotted against eachother, together with the Earth and Moons centres.

Furthermore, the x and y positions as well as velocities are plotted against time as the x-axis. One can see that the graph makes sense, considering the trajectory of the satellite visualised in figure 6 above.

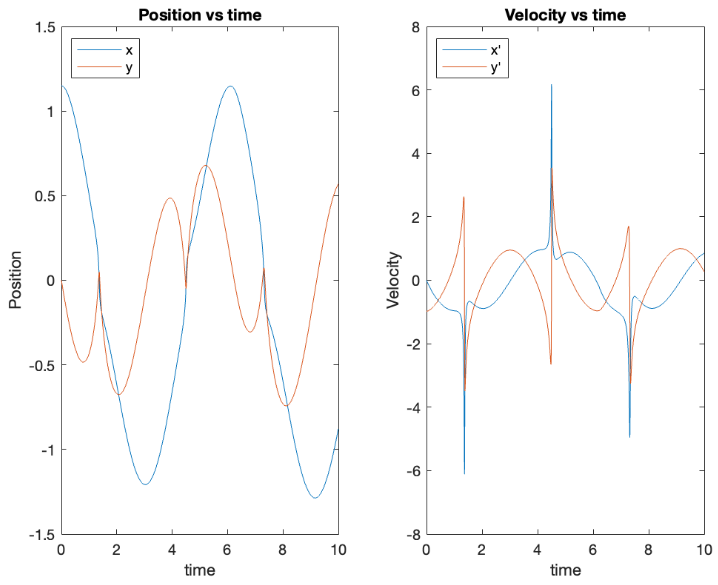

Figure 6: Position vs time and Velocity vs time, both for x and y coordinates.

## b)

In this part we compare the different methods Explicit Euler, Runge-Kutta 3 and Adams-Bashforth 4 when increasing the duration of time. Since the numerical error generally increases with time the number of steps, $N$, were collected, where the stepsize $h = 1/N$. The $N$ to be collected for each method were approximately the least number of steps needed for keeping the error:

$$|\tilde{\mathbf{r}}(T_{end}) - \mathbf{r}(T_{end})| < 0.1.$$

We were provided with a table of exact solutions of **r** at three different points of time: 5, 20 and 40. Here is the given table.

| **r** | t = 5 | t = 20 | t = 40 |
|---|---|---|---|
| x(t) | 0.4681 | -0.2186 | -1.4926 |
| y(t) | 0.6355 | -0.2136 | -0.3339 |

Moreover, here is the table of collected amount of timesteps, where X represent that $N$ was excessively large for the method to catch it within reasonable time.

| Method | t = 5 | t = 20 | t = 40 |
|---|---|---|---|
| Explicit Euler | N = 200000 | N = X | N = X |
| Runge-Kutta 3 | N = 705 | N = 2100 | N = 16500 |
| Adams-Bashforth 4 | N = 1035 | N = 1246 | N = 4400 |

For $t = 20$, $N = 500000$ was used to get the error for Explicit Euler decreased to 0.35, which took about five minutes on our computer, which we consider as "excessively large".

As expected, Adams-Bashforth 4 could be used with smaller values of $N$ than Runge-Kutta 3 or Explicit Euler could, when solving the problem for $t = 20$ or $t = 40$. This is expected, since Adams-Bashforth 4 is a multi-step method. However when solving for $t = 5$, Runge-Kutta 3 obtained a more accurate solution for a smaller value of $N$. Explicit Euler is a first order method which needs very small time steps in order to obtain accurate results.

RK3 is a one-step method, while AB4 is a four-step method. This means that AB4 is computationally much more expensive in each step. Hence, we would prefer using RK3 up to $t = 20$. For larger time intervals than this however, the difference in what value of $N$ results in an accurate solution is large enough between the two methods for AB4 to be about equally expenive in total, since $N_{RK3} \approx 4 * N_{AB4}$ when $t = 40$.

## c)

We now solve the problem using the MATLAB built-in method ode23, for which we verify that the same accuracy as above is satisfied. For $T_{end} = 20$, we get an error of 0.0830, which is within the wanted error span. By checking the length of the time vector, we see that the number of time steps used by ode23 for this solution is 974. Compared to the Adams-Bashforth method, for which we needed 1246 steps to obtain a similar error, one can regard ode23 as more accurate.

The largest and smallest time step, respectively, that ode23 assumes can be extracted by comparing each value in the time vector with its successor. We find that the largest time step is 0.1020 and the smallest is 1.6115e-04 (se figure 7). For Adams-Bashforth 4, every time step is of the size 8.0257e-04, which is only slightly bigger than the smallest time step of the ode23 method.

This seems reasonable, since ode23 is an adaptive method which will take very small steps only when the magnitude of the second derivative is large. Hence, the smallest time steps will be taken around the extreme points of the function. However in other parts of the solution, bigger time steps can still approximate quite well, since the second derivative is very small.

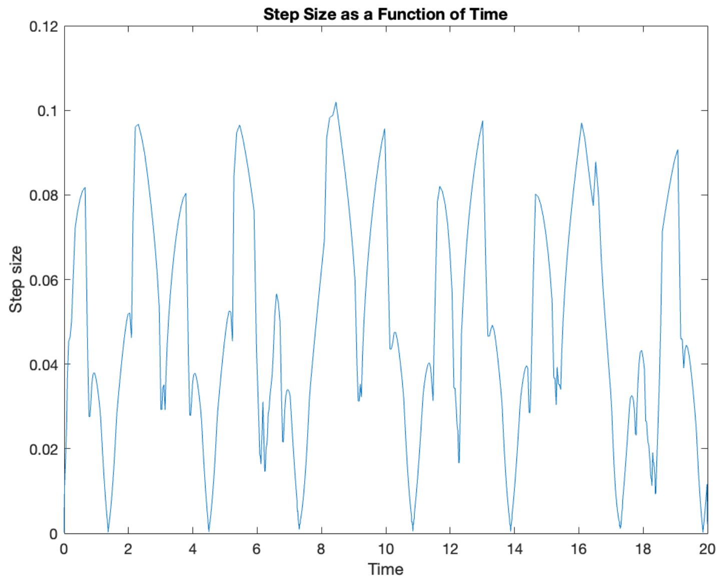

Figure 7: Step size as a function of time when solution is approximated by ode23.

## Problem 3

In this exercise, we consider the Robertson problem which here is modeling the reactions of three chemicals, A, B and C. We solve the problem numerically using the explicit Runge-Kutta 3-method. We are given the following system of three equations:

$$\begin{pmatrix} \frac{dX_A}{dt} \\ \frac{dX_B}{dt} \\ \frac{dX_C}{dt} \end{pmatrix} = \begin{pmatrix} -r_1X_A + r_2X_BX_C \\ r_1X_A - r_2X_BX_C - r_3X_B^2 \\ r_3X_B^2 \end{pmatrix},$$

which describes the evolution of the concentration of the chemicals over time, where $r_i$ are the rates of the three reactions.

The problem is stiff, hence we need a pretty small timestep, $h$, in order to obtain a stable solution. By plotting the solution for different values for $h$, we decided to use $h_0 = 0.0007$. Using a smaller stepsize than this makes no noticable difference in the graph, but using a significantly larger stepsize gives an unstable solution. The solution using Runge-Kutte 3 with $h_0 = 0.0007$ is plotted in figure 8 for $t \in [0, 10]$.

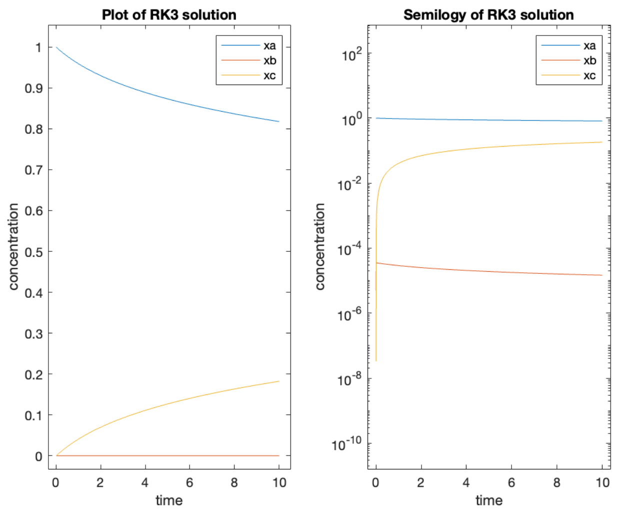

Figure 8: Numerical solution to the Robertson problem using RK3 with $T = 10$ and stepsize $h_0 = 0.0007$.

We now find the jacobi matrix for the right hand side of our system of equations:

$$J = \begin{pmatrix} -r_1 & r_2 X_C & r_2 X_B \\ r_1 & -r_2 X_C - 2r_3 X_B & -r_2 X_C \\ 0 & 2r_3 X_B & 0 \end{pmatrix},$$

and calculate the eigenvalues of $J$ for each timestep.

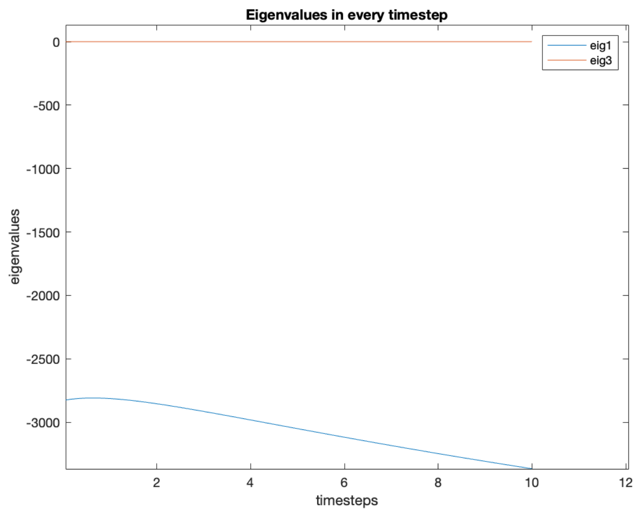

Figure 9: Eigenvalues of the Jacobian for every timestep.

The largest eigenvalue in magnitude at $t = 10$ is -3365.9 (see figure 9). This value should indicate the largest stepsize we can use in order for our numerical method to be stable. We check this by determining whether we are within the stability region for Runge-Kutta 3 or not.

As defined in problem 1, the stability region for RK3 is all $z$ in the left half of the complex plane for which $s(z) = |1 + z + z^2/2 + z^3/6| - 1 \leq 0$. We assign $z_0 = h_0 * (-3365.9)$, where $h_0 = 0.0007$ is the time step we previously mentioned seemed fit for this problem, and get $s(z_0) = -0.2396$. For a larger timestep, eg $h_1 = 0.00085$, $s(h_1 * (-3365.9))$ is instead evaluated to a positive number, suggesting this timestep be too large. Hence, our chosen timestep, $h_0 = 0.0007$, seems reasonable.

As the solution to our system of equations is nonlinear, the stability region changes over time. The largest eigenvalue's magnitude gets more severe as the number of timesteps increases, hence the largest timestep needed for a stable solution gets smaller.

To solve the equation for $T = 1000$, we used stepsize $h = 0.0001$. As suggested by the increasing magnitude of largest eigenvalue of the Jacobian as the number of timesteps increases, we use a smaller timestep here than when only plotting up to $t = 10$. The final position for the three solutions using RK3 for $t \in [0, 1000]$ was $(X_A, X_B, X_C) = (0.2934, 0.0000, 0.7066)$.

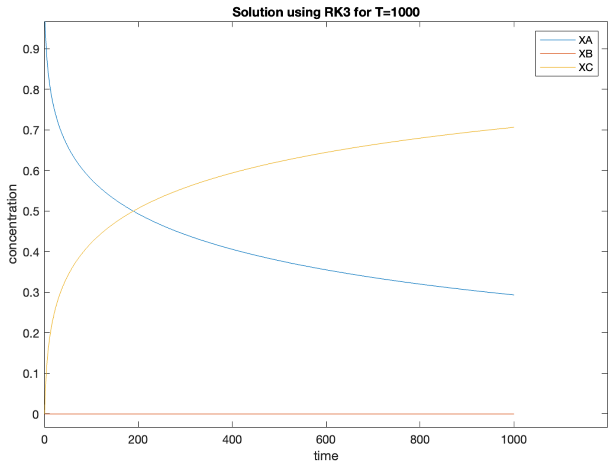

Figure 10: Numerical solution to the Robertson problem using RK3 method with $T = 1000$ and $h = 0.0001$.

If we run this code and save the solution in each step, it takes approximately 31 seconds (using tic-toc function in MATLAB). If we run the code without saving the solution in each step however, it only takes 5 seconds.

Next, we plot the solution to the same equation using Implicit Euler using a stepsize of 1. Since this method is stable, no matter the stepsize, we can use a much larger $h$.

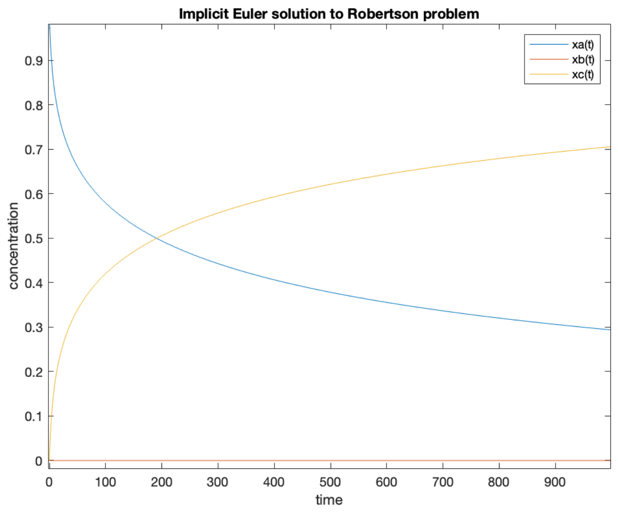

Figure 11: Numerical solution to the Robertson problem using the Implicit Euler method with $T = 1000$ and $h = 1$.

In order to compare the efficiency of the two methods, RK3 and Implicit Euler, we modified both methods so they do not save the values of the vectors in each step. With tic/toc we could then time the method for different values of $h$. Results are reported below.

| Method | h | error | computational time |
|---|---|---|---|
| RK | 0.0001 | 1.6665e-08 | 5.0517 |
| IE | 1 | 4.9344e-04 | 0.0156 |
| IE | 0.1 | 5.0291e-05 | 0.0498 |
| IE | 0.01 | 5.0401e-06 | 0.4463 |
| IE | 0.001 | 5.0413e-07 | 4.3598 |
| IE | 0.0001 | 5.0414e-08 | 43.1440 |

The timing of the methods are influenced by the number of floating point operations per timestep as well as the number of timesteps. In Implicit Euler, we have stable solutions even for large timesteps, which makes the computations fast. However, if we compare IE to RK3 when we have the same small (and stable) timestep for both, RK3 is much faster since it is less computationally expensive in each step.

Also, the errors of the methods are affected by the number of timesteps, since a large timestep creates bigger local truncation errors, which accumulate to a large global error. The order of the method also has a big influence on the error. RK3 is a third order method and will therefore in general produce solutions with a smaller error than IE which is a first order method.

When very accurate solutions are sought, IE needs equally small time steps as RK3 for the problem we are working with. Since IE is more computationally expensive in each timestep, this means IE is slower for such a high level of accuracy. However if less accurate results are sought, we can use IE with larger timesteps and still get a stable solution that is found much quicker. Since RK3 is conditionally stable, the same thing is not possible for that method, and no quicker solution can be computed.

IE is faster than RK if the error level desired is roughly $5.0413e - 07$ or higher. For lower error levels, RK3 is faster.

If the same experiments were made for the problem in part 1, where the Runge-Kutta 3 method was stable for a much larger value of $h$, we would expect RK3 to be a better method even when seeking results of lower accuracy. Although an unconditionally stable method, Impicit Euler would not be needed in order to obtain accurate results in a very short amount of time, since RK3 is capable of producing quicker solutions for this problem, that do not need to have very small time steps.