Homework 1. Numerical methods for ODES

Valentina Zonta

November 30, 2023

1 Exercise 1

I implemented the 2-step Simpson method to solve the test equation

$$y' = -5y \quad , y(0) = 1 \tag{1}$$

with h=0.05 and $t_N = 6$

The 2-step Simpson method follows the formula:

$$y_{n+2} = y_n + \frac{h}{3}\left(f_n + 4f_{n+1} + f_{n+2}\right), \tag{2}$$

where in this particular case $f(t, y) = -5y$.
The code used for this method is the following:

```
function simspon2(c,d)
format long;
    h = 0.05;  % step size
    T = 6;  % final time
    N = T / h;  % number of intervals

    % Initialize arrays
    t = 0:h:T;
    y = zeros(N+1, 1);
    y(1) = 1;
    y(2)=c;



     % Perform the 2-step Simpson's method
    for i = 1:N-1
        y(i+2) = (y(i) + (h / 3) * (-5 * y(i) + 4 * (-5 * y(i+1))) )/(1+(5/3)*h );
    end
```

To iterate a two-step method I needed to compute the second initial value y(1). I computed it in three ways:

- as the exact solution e^{-5h}
- by a 4th order Runge-Kutta method
- with the forward Euler method

I created the Simpson function such that in the input I could insert the second initial condition, found in three different ways as just written above.
The first method is trivial. In the second one, I used the 4-order runge kutta formula

$$y_{n+1} = y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right), \tag{3}$$

where

$$
\begin{aligned}
k_1 &= f(x_n, y_n), \\
k_2 &= f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right), \\
k_3 &= f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right), \\
k_4 &= f\left(x_n + h, y_n + hk_3\right).
\end{aligned}
$$

For the last one I used

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{4}$$

The values of the second initial condition are:

	exact	4RK	FE
y(2)	0.778800	0.778808	0.750000

Using the three different y(2) I plot the error as the difference between the exact and the approximate solution

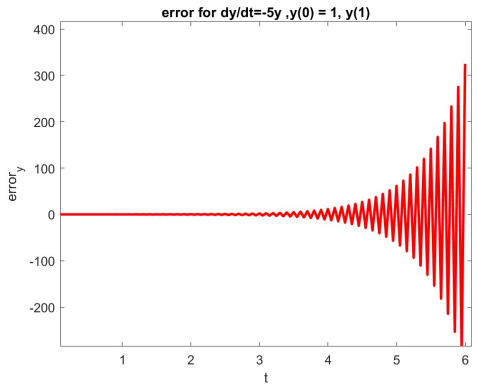

Figure 1: errors obtained with FW

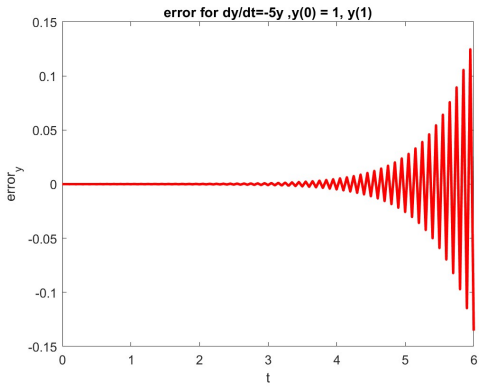

Figure 2: errors obtained with 4RK

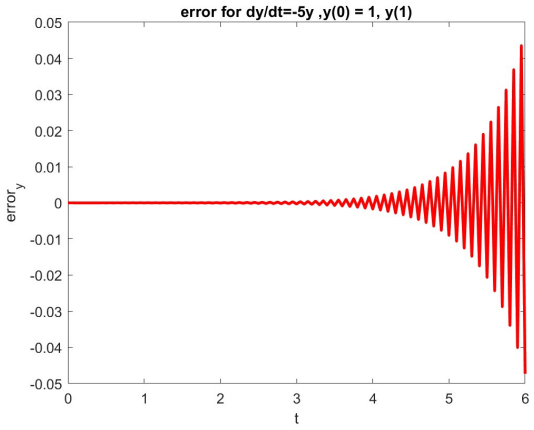

Figure 3: error from real value of y(2)

In conclusion, the initial values y(2) obtained with the three methods are similar for 4RK and real value, (in fact the error's magnitudes are close to each other), and are much larger than the one obtained with FE (10^{3-4} bigger). In all the cases the error diverges implying instability in the numerical solution.

2 Exercise 2

To solve the following equation using the 4-stage Runge Kutta method

$$\frac{dy}{dt} = -10y^2, \quad y(0) = 1 \tag{5}$$

with $t \in (0, 2)$ and $h = 2^{-k} \quad k = 5, \dots 10$ I created a function that gives back $y(t)$ and calculate the error at the final time T=2. The following plot creates a log-log plot of this error as a function of the number of steps

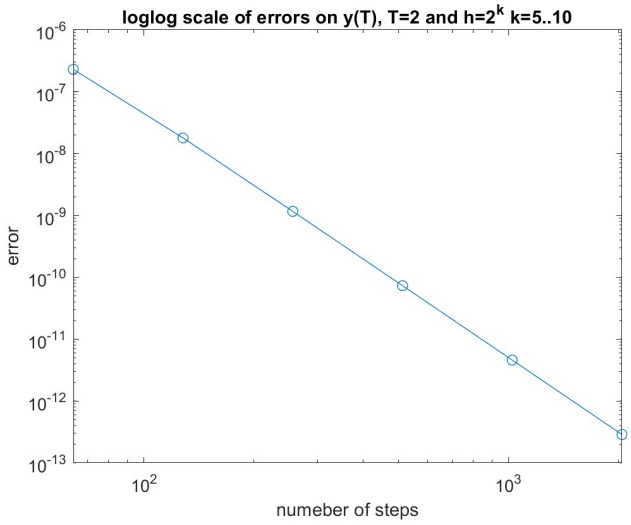

Figure 4: loglog plot of error

I also provide a table with the values of h and the error in each row:

h	errors	Error Ratio(errors[i] / errors[i-1])	Step Size Ratio (h[i] / h[i-1])
0.03125	2.2918×10^{-7}	-	-
0.015625	1.7858×10^{-8}	0.078	0.5
0.0078125	1.1602×10^{-9}	0.065	0.5
0.00390625	7.3129×10^{-11}	0.063	0.5
0.001953125	4.5796×10^{-12}	0.063	0.5
0.0009765625	2.8638×10^{-13}	0.063	0.5

The reduction in error as h decreases confirms the expected behavior of the RK4 method, where decreasing the step size results in increased accuracy. The ratios between consecutive errors and step sizes show a consistent pattern which follows the order of convergence of the error $O(h^4)$ ($\Delta h = \frac{1}{2} \to \frac{1}{16} = 0.0625 \sim$ error ratio in table). Moreover, the plot shows that the errors follow a double logarithmic trend(linear in the log-log plot) with slope 4 implying that the errors follow scale oh h^4.

3 Exercise 3

We assume to use the 4-stage Runge Kutta method and computed y^h_{n+1}, $y^{h/2}_{n+2}$ two approximate $y(tn + h)$. To develop a formula for the Richardson extrapolation we shall start from its definition:
Let $A_0(h)$ be an approximation of $A^*(exactvalue)$ that depends on a positive step size h with an error formula of the form

$$A^* = A_0(h) + a_0 h^{k_0} + a_1 h^{k_1} + a_2 h^{k_2} + \dots \tag{6}$$

In this notation $O(h^{k_i})$ represents the truncation error of the $A_{i(h)*}$ approximation such that $A^* = A_i(h) + O(h^{k_i})$.

By assumption of the homework, the local truncation error is $C_n(h^5)$ with C_n depending on the derivates of y. Therefore we can write:

$$y^* = y_{n+1}^h + C_n(h^5) + O(h^6) \qquad y^* = y_{n+2}^{h/2} + 2C_n\left(\frac{h}{2}\right)^5 + O(h^6) \tag{7}$$

where the second equation presents a factor 2 in front of C_n because we need two steps of length $\frac{h}{2}$ to get to $= y_{n+2}^{h/2}$.
If we treat the C_n from both equations close to each other, we can solve the system of the previous equation with the unknown quantity y^*. From the second equation, we get

$$C_n = \frac{16}{h^5}\left(y^* - y_{n+2}^{h/2}\right) \tag{8}$$

and substituting it into the first we obtain

$$y^* = y_{n+1}^h + \left(y^* - y_{n+2}^{h/2}\right)\frac{16h^5}{h^5}. \tag{9}$$

Therefore the final result for the exact value of y is:

$$y^* = \frac{-y_{n+1}^h + 16y_{n+2}^{h/2}}{15}. \tag{10}$$

It is also possible to estimate the truncation error from this result using 8

$$C_n h^5 = \frac{16}{15}(y_{n+1}^h + y_{n+2}^{h/2}). \tag{11}$$

In conclusion, applying Richardson extrapolation to the Runge-Kutta method we obtain a more accurate numerical solution. This is acquired using multiple solutions with different step sizes and combining them to reduce the error inherent in the original method.

4 Exercise 4

Let's obtain the BDF2 formula (k = 2) and find its local truncation error. We can write an ODE as

$$y'(t) = f(t, y(t)) \quad y(t_0) = y_0 \tag{12}$$

we can generalize the previous formula as

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\beta_k f(t_{n+k}, y_{n+k}) \quad \beta_{k-1}, \ldots, \beta_0 = 0 \tag{13}$$

For the backward differentiation formula, we need to interpolate the function $y(t)$, instead of f as for forward formulas, using the points $(t_n, y_n), \ldots, (t_{n+k}, y_{n+k})$. We can obtain a Polynomial $P(t)$ using the Lagrange interpolation formula, and approximate $y'(t)$ with $P'(t)$.

For $k = 2$ we have to interpolate 3 points, so from the general Lagrange interpolation formula:

$$P_j(x) = y_j \prod_{k=0, k\neq j}^{n} \frac{x - x_k}{x_j - x_k} \tag{14}$$

.

we get the following polynomial

$$P(t) = y_n \frac{(t - t_{n+1})(t - t_{n+2})}{2h^2} + y_{n+1}\frac{(t - t_n)(t - t_{n+2})}{-h^2} + y_{n+2}\frac{(t - t_n)(t - t_{n+1})}{2h^2} \tag{15}$$

We now derive eq. 15

$$P'(t) = \frac{y_n}{2h^2}[(t - t_{n+1}) + (t - t_{n+2})] + \frac{y_{n+1}}{-h^2}[(t - t_n) + (t - t_{n+2})] + \frac{y_{n+2}}{2h^2}[(t - t_n) + (t - t_{n+1})] \tag{16}$$

As said earlier we want to find $P'(t_{n+k})$, so in this case $P'(t_{n+2})$, knowing that $t_{n+k} - t_{n+k-1} = h$

$$P'(t_{n+2}) = \frac{y_n}{2h} - \frac{2y_{n+1}}{h} + \frac{3y_{n+2}}{2h} \tag{17}$$

It is possible now to write 13 in this particular case

$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_{y_n} = \frac{2}{3}hf(t_{n+2}, y_{n+2}). \tag{18}$$

Recalling the interpolation error

$$E(t) = \frac{(t-t_k)\dots(t-t_n)f^{k+1}(\eta(t) =}{(k+1)!} \tag{19}$$

to obtain the local truncation error it is necessary to calculate $|E'(t)|$ in its maximum. So calculating the derivate for $k = 2$:

$$E'(t) = \frac{1}{6}\big([(t-t_{n+1})(t-t_n)f^3(\eta(t))] + [(t-t_{n+2})(t-t_n)f^3(\eta(t))] + [(t-t_{n+1})(t-t_{n+2})f^3(\eta(t))] + \\ + [(t-t_n)(t-t_{n+1})(t-t_{n+2})f^4(\eta(t))(\eta')]\big)$$

If f^4 is neglected,

$$\begin{aligned} E'(t) &= \frac{f^3(\eta(t))}{6}([(t-t_{n+1})(t-t_n)] + [(t-t_{n+2})(t-t_n)] + [(t-t_{n+1})(t-t_{n+2})] \\ &= \frac{f^3(\eta(t))}{6}(3t^2 - t*2*(t_n + t_{n+1} + t_{n+2}) + t_nt_{n+2} + t_{n+1}t_{n+2} + t_nt_{n+1}) \end{aligned}$$

that is a second order plynimial in t, with the constant $at^2, a > 0$, so geometrically a convex parabola. Therefore the maximum is in the extreme of the domain.

Choosing $k = 2$ the extreme are in n and $n + 2$, so substituting them into the previous equation:

$$E'(t_n) = \frac{h^2}{3}f^{(3)}(\eta(t_n)) \qquad E'(t_{n+2}) = \frac{h^2}{3}f^{(3)}(\eta(t_{n+2})) \tag{20}$$

To prove that BDF2 is absolutely stable in the real interval $\bar{h} \in (-\infty, 0)$ we shall give the definition of region of absolute stability:
A linear multistep method is called stable for a given $\bar{h}$ if and only if all the roots $r_s(\bar{h})$ of the polynomial $\pi(z, \bar{h})$ satisfy

$$|r_s|, \quad s = 1, \dots, k. \tag{21}$$

The region in the complex plane (or the interval) where this occurs is called the region of absolute stability. In our case the polynomial and its roots are

$$t^2(3 - 2\bar{h}) - 4t + 1 = 0 \qquad t_{1,2} = \frac{2 \pm \sqrt{1 + 2\bar{h}}}{3 - 2\bar{h}} \tag{22}$$

so we have to find the region where $|t_{1,2}| < 1$ in function of $\bar{h}$. We can divide it into two cases, whether the solutions are real or complex.

- If the solutions are real the intersection of two intervals is $\bar{h} > 4 \cup -\frac{1}{2} \leq 0 \cap \bar{h} > \frac{3}{2} \cup -\frac{1}{2} \leq \frac{3}{2}$. Therefore if we are checking the stability in $(-\infty, 0)$, our solution exists for $\bar{h} \leq 0$, so in all the interval.
- If the solutions are complex, so $1 + 2h < 0$ we can write

$$t_{1,2} = \frac{|2 \pm i\sqrt{-1 - 2\bar{h}}|}{|3 - 2\bar{h}|} = \frac{\sqrt{3 - 2\bar{h}}}{|3 - 2\bar{h}|} \tag{23}$$

  and the solution exist for $\bar{h} < 1$ so in all our interval.

5 Exercice 5

We want to solve the linear system of ODE's

$$y'(t) = -Ay(t)$$
$$y(0) = [1111\dots1]^T$$

with three different methods.
In all cases A is generated by the commands:

```
    nx = 1 0 0;
G = numgrid ( ' S ' , nx ) ;
A = d el s q (G)  ( nx1)^2 ;
```

and $t \in (o, o, 1]$. The exact solution at final time is

$$y = exp(-0.1A)y_0 \tag{24}$$

From theory, we know that we can rewrite an ODE's system using a diagonal matrix D and a matrix V whose columns are the eigenvectors of A. Calling $z(t) = V^{-1}y(t)$ the system can be written as

$$z_i'(t) = \lambda_i z_i(t) \tag{25}$$

where λ_i are the eigenvalues of the matrix A.
Studying the stability of runge kutta methods and considering the model problem

$$y' = \lambda y \quad , \qquad y(0) = y_0 \tag{26}$$

we define a new variable $\bar{h} = h\lambda$. The method of Runge Kutta of R-stage results in a form like

$$y_{n+1} = A_r(\bar{h})y_n \tag{27}$$

and is said to be absolutly stable if $| = A_R(\bar{h})| < 1$.

So to compute the interval of absolute stability of the 4th order Runge-Kutta method for this problem, having for each element of the system a model problem-like equation (26 $\sim$ 25), we simply have to solve

$$|A_4(\bar{h})| = |1 + \bar{h} + \frac{\bar{h}^2}{2} + \frac{\bar{h}^3}{6} + \frac{\bar{h}^4}{24}| < 1 \tag{28}$$

with $\bar{h} = h\lambda^*$ with λ^* the largest modulus eigenvalues of A.

Computing λ^* with `lambda = -eigs(A,1,'lm')` and calculating the inequality we find that the interval of absolute stability is $h \in (0, 3.546 \cdot 10^{-5})$.

Then I solved the problem using three methods (ode45, Crank Nicolson, BDF3) .
For the first one, I solved the problem by the function ode45 and computed the infinity modulus errors concerning the exact value.
For the second one (CN) I have written the method applied to our system more conveniently, to apply the Conjugate Gradient method. From

$$y_{n+1} = y_n + \frac{h}{2}(-Ay_n - Ay_{n+1}) \tag{29}$$

to

$$(I + \frac{h}{2}A)y_{n+1} = (I - \frac{h}{2}A)y_n. \tag{30}$$

I computed this last system using `pcg` function in matlab for three different step size $h = [10^{-3}10^{-4}10^{-5}]$ and tolerance 10^{-3}. Finally, I computed the infinity norm errors.

For the last method (BDF3) I have written, as in the previous case, the system in a convenient way. From

$$y_{n+3} - \frac{18}{11}y_{n+2} + \frac{9}{11}y_{n+1} - \frac{2}{11}y_n = -\frac{6}{11}hA \tag{31}$$

to

$$y_{n+3}(I + \frac{6}{11}A) = \frac{18}{11}y_{n+2} - \frac{9}{11}y_{n+1} + \frac{2}{11}y_n. \tag{32}$$

To solve this three-step problem I needed two other initial conditions. I simply used y(1) and y(2) already calculated in CN. I then applied the pcg function for the stemps size $h = [10^{-3}10^{-4}10^{-5}]$, tolerance 10^{-3} and computed the errors.
For all three methods, I saved CPU time. The results are summed up in the following table

method	h	errors	CPU time	num step
ode45	*	1.155e-05	10.80	9445
CN	10^{-3}	3.783e-05	0.39	100
CN	10^{-4}	1.397e-07	1.38	1000
CN	10^{-5}	1.396e-09	11.89	10000
BDF3	10^{-3}	3.727e-07	0.24	100
BDF3	10^{-4}	3.774e-10	1.37	1000
BDF3	10^{-5}	1.651e-12	10.87	10000

The methods become more precise as h gets smaller as expected .
For CN as we decrease the step size h, the error generally decreases quadratically $O(h^2)$.
The BDF3 Method (Third-order Backward Differentiation Formula) specifically is a third-order accurate method. As the step size decreases, the error for BDF3 diminishes more rapidly, typically with cubic convergence, as we see from the results apart from $h = 10^{-5}$.
The bdf3 method exhibits faster error reduction as compared to lower-order methods. It tends to have higher-order convergence, leading to quicker reductions in error as the step size decreases.

6 Exercise 6

We were given the Lotka-Volterra equations

$$\frac{dx}{dt} = x(t)(\alpha - \beta y(t)),$$
$$\frac{dy}{dt} = y(t)(\gamma x(t) - \delta),$$

with initial conditions:

$$x(0) = x_0, \quad y(0) = y_0$$

I solved the Lotka-Volterra equation with parameters $\alpha = 0.2$, $\beta = 0.01$, $\gamma = 0.004$, $\delta = 0.07$ by a 4th order RK method using as initial conditions $x_0 = 19$, $y_0 = 22$, $t_0 = 0$, $T = 300$ and a step size $h = 10^{-3}$.

To compute the problem I simply used the RK4 method defined in 3 on the two variables that are dependent on each other, so for every cycle of time I had to compute both x and (not independently) as follows:

```
for n = 1:N
K1x = f(x_vals(n), y_vals(n));
K1y = g(x_vals(n), y_vals(n));

K2x = f(x_vals(n) + h * K1x / 2, y_vals(n) + h * K1y / 2);
K2y = g(x_vals(n) + h * K1x / 2, y_vals(n) + h * K1y / 2);

K3x = f(x_vals(n) + h * K2x / 2, y_vals(n) + h * K2y / 2);
K3y = g(x_vals(n) + h * K2x / 2, y_vals(n) + h * K2y / 2);
```

```
    K4x = f(x_vals(n) + h * K3x, y_vals(n) + h * K3y);
    K4y = g(x_vals(n) + h * K3x, y_vals(n) + h * K3y);

    x_vals(n + 1) = x_vals(n) + h * (K1x + 2 * K2x + 2 * K3x + K4x) / 6;
    y_vals(n + 1) = y_vals(n) + h * (K1y + 2 * K2y + 2 * K3y + K4y) / 6;
end
```

The results for prey and predator are given in the following plot.

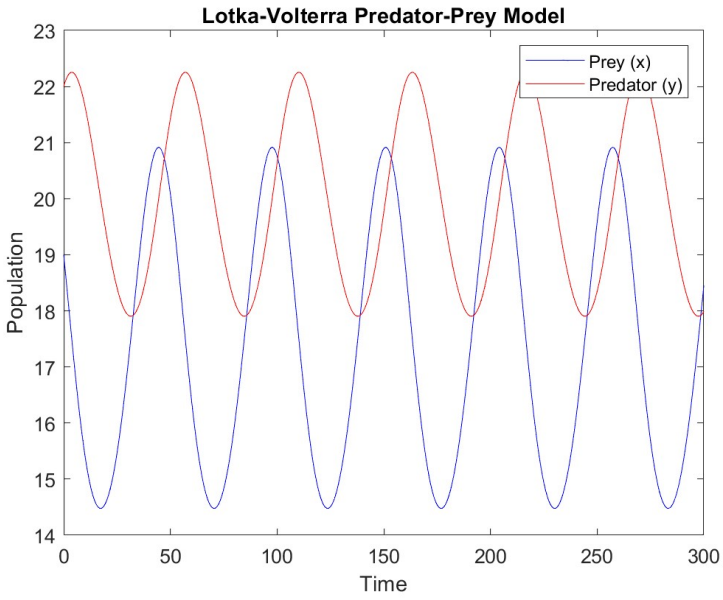

Figure 5: Lotka-Volterra results