University of Costa Rica
School of Mathematics
MA-501 Numerical Analysis
Homework 4

Juan Ignacio Padilla B.
ID: B55272
Prof. Juan Gabriel Calvo
January 29, 2026

Homework 4

Problem 1.

a) Write a function that takes as input $m \in \mathbb{N}$ and returns the matrix $A \in \mathbb{R}^{m \times m}$ given by

$$A = \begin{bmatrix} 1 & & & & & 1 \\ -1 & 1 & & & & 1 \\ -1 & -1 & 1 & & & 1 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ -1 & -1 & \dots & -1 & 1 & 1 \\ -1 & -1 & \dots & -1 & -1 & 1 \end{bmatrix},$$

where the entries on the diagonal and the last column are 1, and the entries below the diagonal are -1.

Solution The function is attached in the documents.

b) For $m = 60$, calculate the growth factor of A,

$$\rho(A) = \frac{\max |u_{ij}|}{\max |a_{ij}|},$$

and the norm $||A - LU||_2$. Comment on the results. You may use the `lu` command.

Solution: The values were calculated using `MATLAB` to obtain

$$\rho(A) = 5.76 \times 10^{17}$$

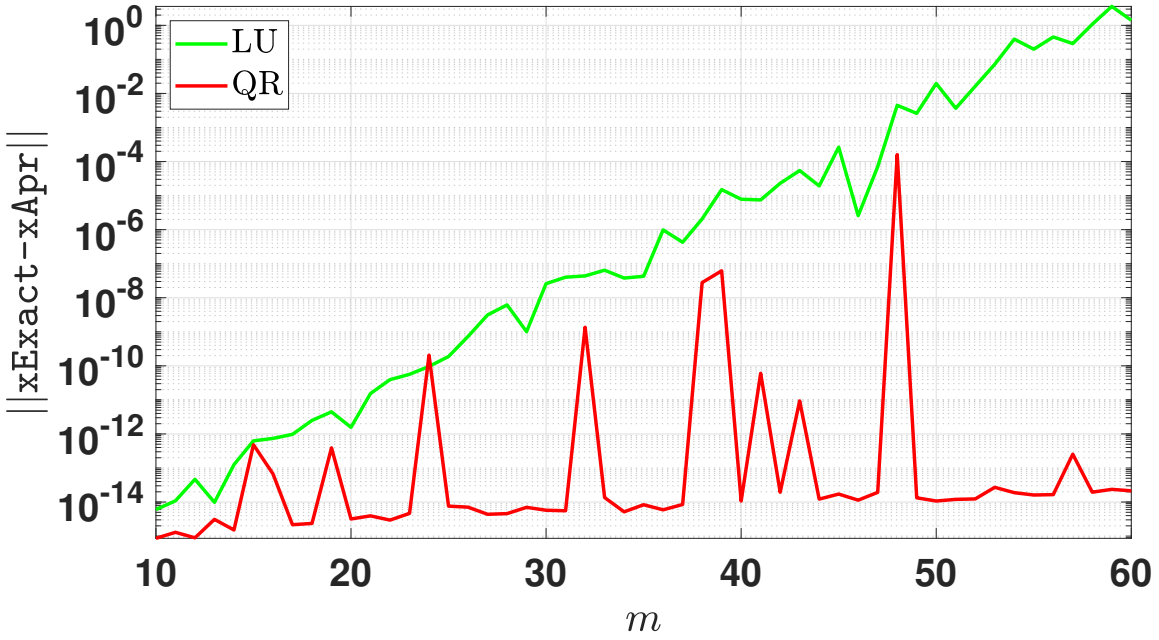
$$||A - LU||_2 = 72.1526$$

It can be observed that the error obtained in the LU approximation of A is very high. This is precisely due to the high growth factor of A (in fact, it is the matrix with the highest growth factor in $\mathbb{R}^{60 \times 60}$).

c) For each `m=10:60`, define `xExact = rand(m,1)` and compute `b=A*xExact`. Solve the system `A*xApr = b` using LU factorization and QR factorization (You may use the `lu` and `qr` commands). Obtain a graph of $||$`xExact-xApr`$||_2$ as a function of m. Justify the behavior of the graph.

Solution: When implementing the solution in `MATLAB`, we obtain graph 1.

Gráfica 1



Once again, the dependence on the growth factor ρ of the LU method can be observed, since its error grows rapidly as m grows. Meanwhile, although the QR method shows error spikes, there is no real growth that depends on m, since it was theoretically demonstrated that the stability of the method does not depend on the growth factor of the matrix. **Note:** These spikes appeared in all runs of the script; it should be noted that the matrices we are evaluating are random.

Problem 2

(Adaptive Runge-Kutta) In class we discussed the explicit fourth-order Runge-Kutta method, given by the scheme

Scheme 1

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

For that case we assumed that h is constant in each iteration. Next we describe an adaptive method, where at each step it is determined whether the value h should be increased or decreased, according to local error considerations. We want to solve the problem

$$\begin{cases} y' = f(t,y), & t \in [0, T_{\max}] \\ y(t_0) = y_0 \end{cases}$$

Using the explicit fourth-order Runge-Kutta method, with variable *step size*.

The program includes three parameters: $h_{\max}$ (the maximum possible value of h, for stability considerations), *tol* (the prescribed tolerance) and h_0 (initial value of h).

At each iteration k, the local error Δ is estimated as follows. First, the approximation y_1 (at time $t_k + h$) is calculated using one Runge-Kutta iteration with increment h. Then, an approximation y_2 (at time t_k+h) is calculated using two Runge-Kutta iterations with increment $\frac{h}{2}$ each. Thus we take $\Delta = y_2 - y_1$.

The value of h is modified as follows. Calculate

$$\tilde{h} = \begin{cases} Sh|\frac{tol}{\Delta}|^{0.20}, & \text{if } |\Delta| \leq tol \\ Sh|\frac{tol}{\Delta}|^{0.25}, & \text{if } |\Delta| > tol \end{cases}$$

where S is a *safety* factor close to 1; in our case take $S = 0.9$. Then, we take $h_{new} = \min\{h_{\max}, \tilde{h}\}$. Finally, if $|\Delta| < tol$, we take the value $y_{k+1} = y_2 + \Delta/15$; otherwise the cycle is repeated. Note that in any case we continue with the value h_{new}.

a) Write a function that implements the adaptive method described above. The inputs should be $f, y_0, t_0, T_{\max}, h_0, h_{\max}, tol$. (Suggestion: to calculate y_1 and y_2 you can use the function discussed in class that implements scheme 1).

Solution: The function `RKadaptive` is attached.

b) Use the previous function for the case $f(t) = \log(t + 0.01) + 1/(t - 4), y_0 = 0.8, t_0 = 0$, $T_{\max} = 3.99$. Take $h_0 = 10^{-5}$, $h_{\max} = 10^{-2}$, $tol = 10^{-12}$. Plot h as a function of time (on an appropriate scale). Comment on the observed variations in h. Also plot the error $|y(t) - \tilde{y}(t)|$ as a function of t.

Solution: First we calculate the general solution of the equation

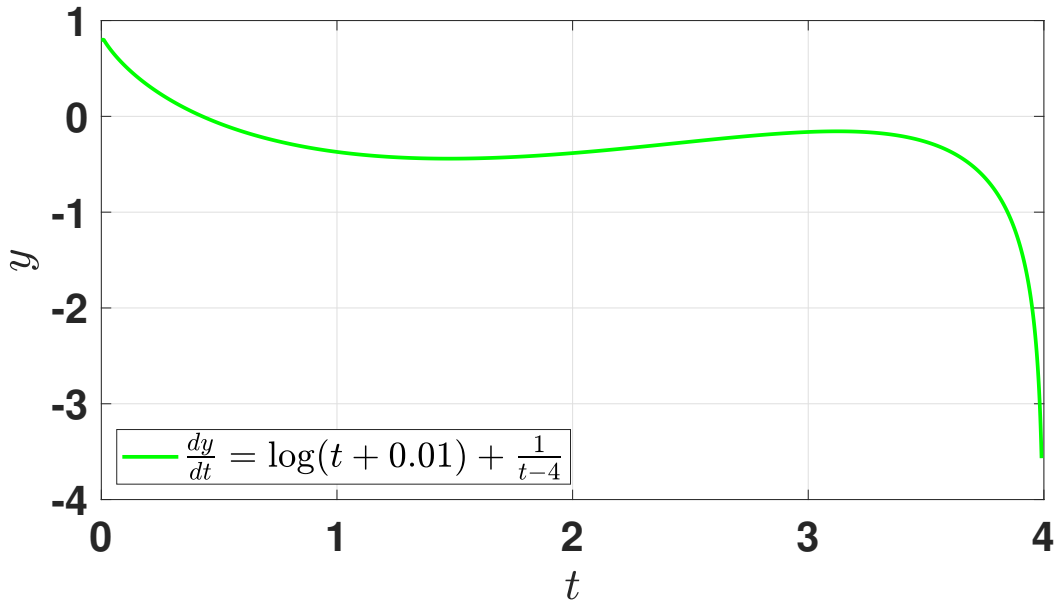
$$\begin{aligned} y' &= \log\left(1 + \frac{1}{100}\right) + \frac{1}{t-4} \\ \Rightarrow y &= \int \log\left(1 + \frac{1}{100}\right) + \frac{1}{t-4} dt \\ \Rightarrow y &= \left(t + \frac{1}{100}\right)\log\left(t + \frac{1}{100}\right) - t - \frac{1}{100} + \log|4 - t| + C \end{aligned}$$

And solving for the initial condition $y(0) = 0.8$, we have $C \approx -0.5302426593$.

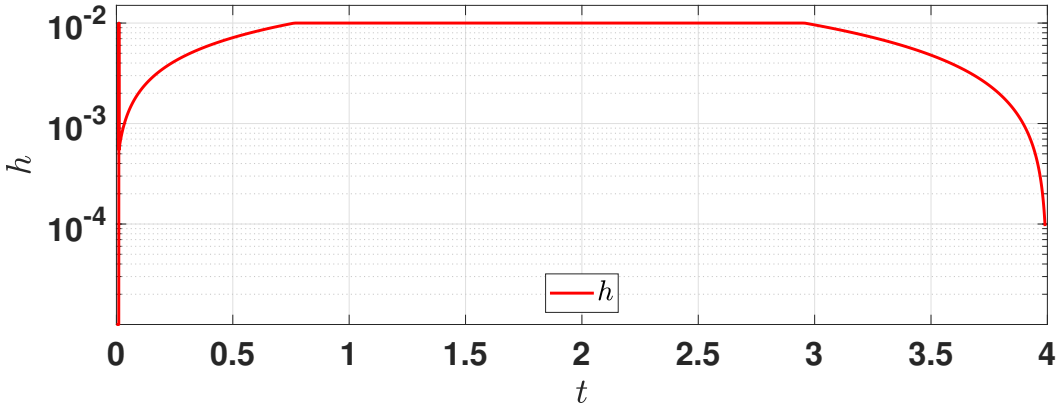
Then, when applying the function from part a), we obtain Graphs 2, 3 and 4. These correspond to a graph of the solution to the equation (obtained using the adaptive method), the step size h, and the error between the general solution and the approximation, respectively.

In the first graph, one can appreciate that at least roughly, the solution seems to approach the expected one (the general solution can be entered into any function plotter), which is a good indicator.

Gráfica 2

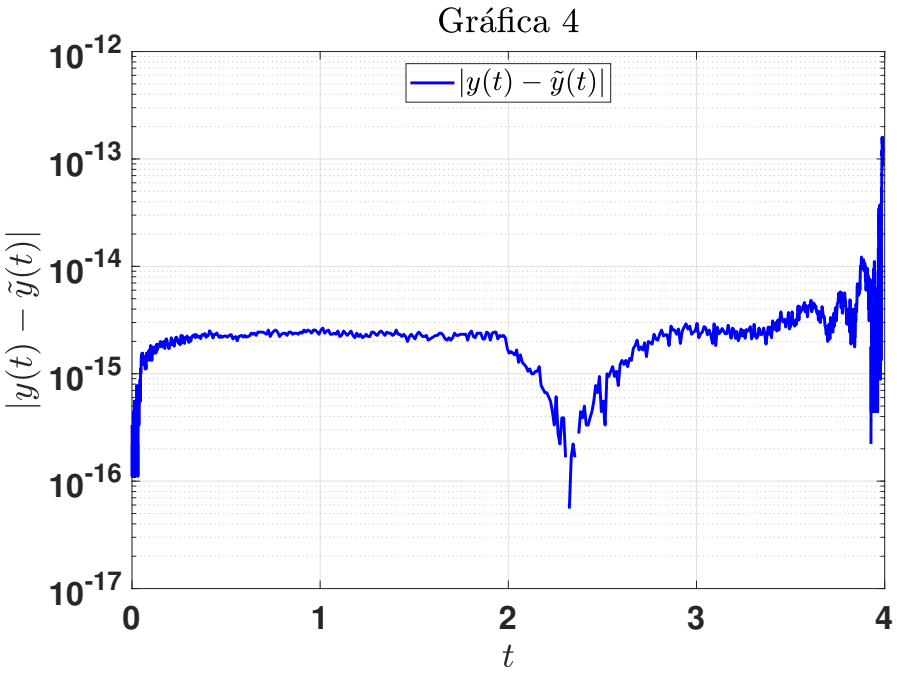


Gráfica 3



Regarding Graph 3, the first thing to note is an abrupt change in the step size in the first iterations. This tells us that our initial step is too small for the adaptive method, so it is quickly readjusted. Subsequently, this value stabilizes for most of the iterations. Finally, approaching the singularity of the function at $t = 4$, the step size must be adjusted again, due to the anomalous behavior of the function near 4.

Finally, regarding the quality of the approximation, in Graph 4 one can note that the approximation is extremely precise, even with a very small number of iterations (934). This reveals the power of adaptive methods, as they ensure convergence and computational economy.



Problem 3

Consider the differential equation given by the system

$$\begin{cases} \frac{\partial^2 x}{\partial t^2} = x + 2\frac{\partial y}{\partial t} - u_c \frac{x+u}{((x+u)^2+y^2)^{3/2}} - u\frac{x-u_c}{((x-u_c)^2+y^2)^{3/2}} \\ \frac{\partial^2 y}{\partial t^2} = y - 2\frac{\partial x}{\partial t} - u_c \frac{y}{((x+u)^2+y^2)^{3/2}} - u\frac{y}{((x-u_c)^2+y^2)^{3/2}} \end{cases}$$

where $x(t), y(t)$ are functions $x, y : [0, \infty) \to \mathbb{R}$. This system models the two-dimensional motion of a satellite, where $(x(t), y(t))$ represents its location at time t. In this case $u = 0.012277471$ and $u_c = 1 - u$ are given constants. Also consider the initial conditions of position and velocity at $t = 0$ given by $x(0) = 0.994, y(0) = 0, x'(0) = 0, y'(0) = -2.001585106$. We know a priori that the motion has period $T = 17.0652166$.

a) Let $z = x', w = y'$ and obtain a first-order system of differential equations with four equations of the form $\boldsymbol{u}' = \boldsymbol{f}(t, \boldsymbol{u}(t))$, $\boldsymbol{u}(0) = \boldsymbol{u}_0$.

Solution: Let us call:

$$F(t) = -u_c \frac{x+u}{((x+u)^2+y^2)^{3/2}} - u\frac{x-u_c}{((x-u_c)^2+y^2)^{3/2}}$$

$$G(t) = -u_c \frac{y}{((x+u)^2+y^2)^{3/2}} - u\frac{y}{((x-u_c)^2+y^2)^{3/2}}$$

Then making the change of variables $z = x'$, $w = y'$, we obtain the system

$$\begin{cases} x' = z \\ y' = w \\ z' = x + 2w + F(t) \\ w' = y - 2z + G(t) \end{cases}$$

Then take

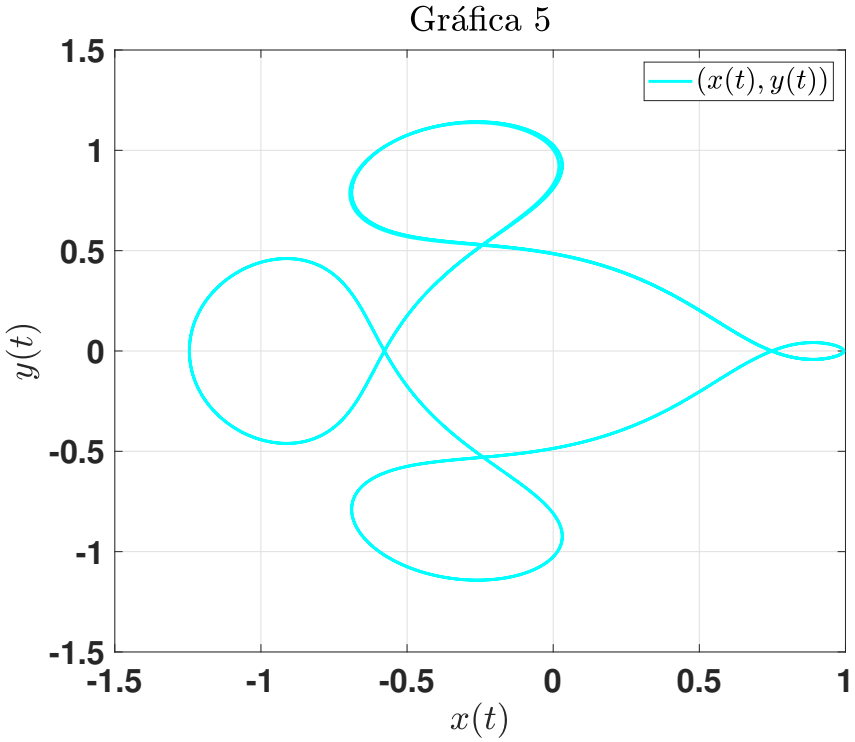
$$\boldsymbol{u} = (x, y, z, w)$$

$$\boldsymbol{f}(t, \boldsymbol{u}(t)) = \begin{pmatrix} z(t) \\ w(t) \\ x(t) + 2w(t) + F(t) \\ y(t) - 2z(t) + G(t)) \end{pmatrix}$$

and $\boldsymbol{u}_0 = (x(0), y(0), x'(0), y'(0))$, to obtain the desired form.

b) Solve the obtained system using the adaptive algorithm from question 3. Use $T_{\max} = 40, h_0 = 10^{-5}, h_{\max} = 10^{-2}, tol = 10^{-6}$. Plot the parametric curve $\{(x(t), y(t)), t \in [0, T_{\max}]\}$. Compare the values $(x(0), y(0)), (x(T), y(T))$ and $(x(2T), y(2T))$. Comment on your results.

Solution: When implementing the algorithm for the application problem, we obtain figure 5, which describes the trajectory



It can be observed that the approximation is quite good; numerically, when evaluating at the periods we obtained that

$$X(0) = 0.994000000000000 \qquad Y(0) = 0$$
$$X(T) = 0.993208458224920 \qquad Y(T) = -0.004486955982549$$
$$X(2T) = 0.994106192301190 \qquad Y(2T) = -1.272527371227908 \times 10^{-4}$$

Therefore the results are stable as t advances.

Note: This run took about 4100 iterations.

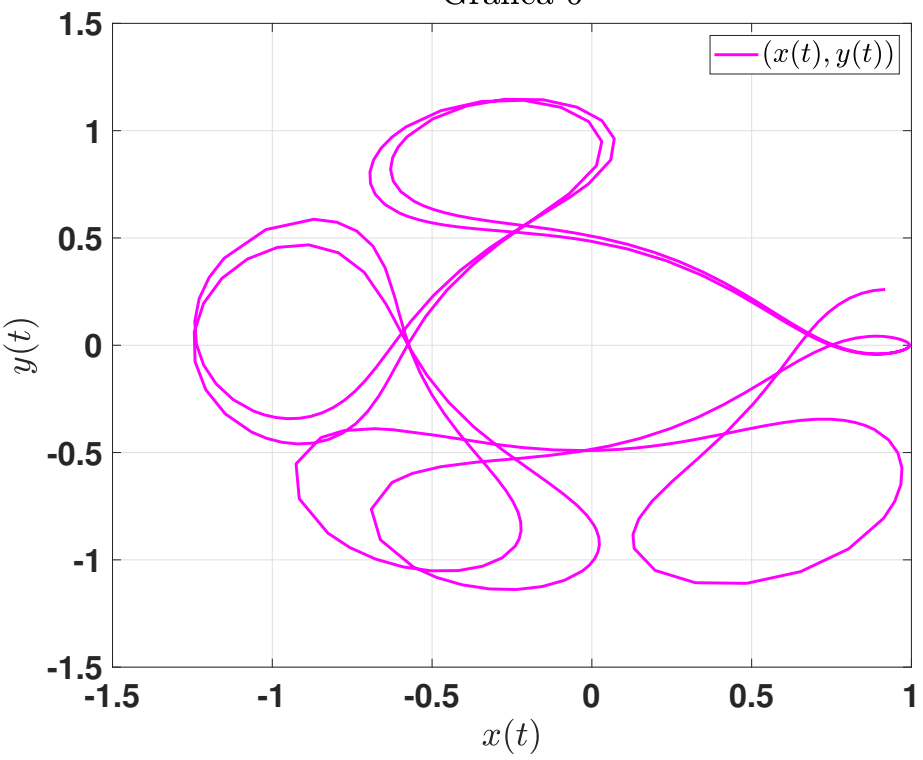
c) Now solve the system using the command

```
[t,u] = ode113(f,[0,40],u0).
```

Plot the parametric curve $\{(x(t), y(t)), t \in [0, T_{\max}]\}$. What do you observe in this case?

Solution: When implementing the `MATLAB` algorithm for the application problem, we obtain figure 6, which describes the trajectory

Gráfica 6



This time we have

$$X(0) = 0.994000000000000 \qquad Y(0) = 0$$
$$X(T) = 0.988300480980110 \qquad Y(T) = 0.012190987212319$$
$$X(2T) = 0.654532855410150 \qquad Y(2T) = -0.349846602949540$$

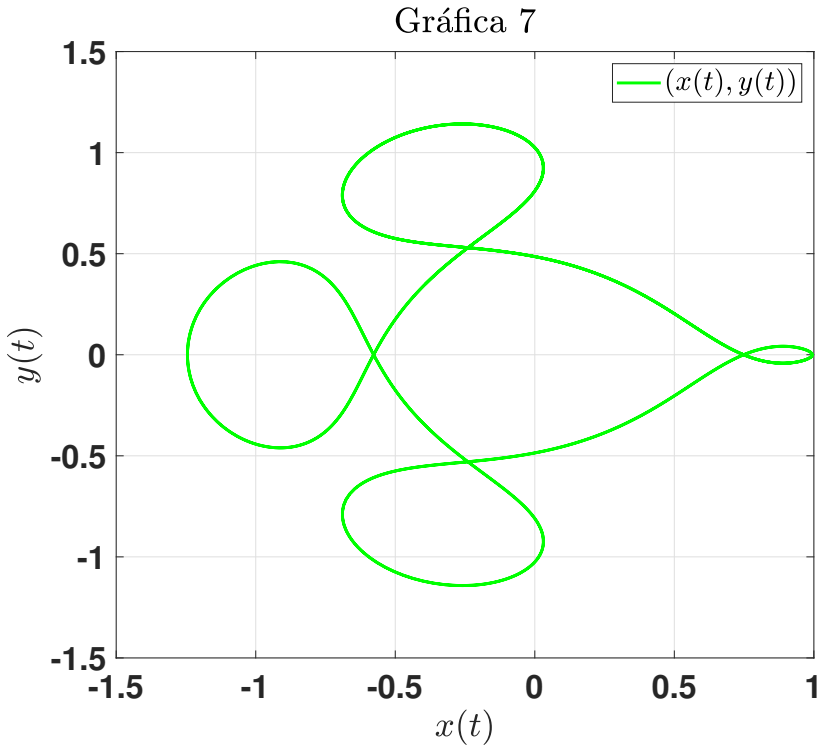
From which we can see that there is a stability problem with the solution, after the second period. However, it is important to highlight that this method requires **far fewer iterations** than the previous one, since it achieved similar results (at least initially), with about 10% of the iterations needed in the previous one.

d) Define the options vector

```
options = odeset('RelTol',1e-10,'Stats','on','AbsTol',1e-10).
```

Now execute the command `[t,u] = ode113(f,[0,40],u0,options)`. Plot the parametric curve $\{(x(t), y(t)), t \in [0, T_{\max}]\}$. What do you observe in this case?

Solution: In this case, modifying the options of the internal `MATLAB` code, we obtain Graph 7.



In which one can see a completely stable solution, which respects the expected period T. Numerically we obtained that

$$X(0) = 0.994000000000000 \qquad Y(0) = 0$$
$$X(T) = 0.993996136853638 \qquad Y(T) = -2.864376096295169 \times 10^{-7}$$
$$X(2T) = 0.993996136853638 \qquad Y(2T) = 4.696898262576229 \times 10^{-5}$$

Therefore we can conclude that this method is optimal for this type of problem, since it did not need as many iterations (it needed 1915), as the manually programmed one (which still offered computational benefits). Furthermore, with few iterations, it offered precise and stable approximations to our initial value problem.