MSAS – Assignment #1: Simulation

Giovanni Chiarolla, 964831

1 Implicit equations

Exercise 1

Given the function $f(x) = \cos x - x$, guess a and b such that $f(a)f(b) < 0$. Find the zero(s) of f in $[a, b]$ with 1) the bisection method, 2) the secant method, 3) the regula falsi method. All solutions must be calculated with 8-digit accuracy. Which method requires the least number of function evaluations? Report the computational time required by each method.

Exercise 1 is about finding the root of the function $f(x)$ using the Bisection method, the Secant method, and the Regula - Falsi method. The three methods have been implemented in a Matlab code that defines an initial range for the root search between a = 0.5 and b = 1.0. The solutions are listed in Tab.1 with an accuracy of 8 digits. The Regula-Falsi and the Secant methods return the same value for the root, while the Bisection method differs from the others by 1×10^{-8}. Tab.2 shows the number of function evaluations required for each method. The bisection method

Table 1: Root of $f(x)$.

Method	Root Value
Bisection	0.73908512
Regula-Falsi	0.73908513
Secant	0.73908513

requires 50 function evaluations, which is the highest number among the three methods, since it relies only on the assumption that the function $f(x)$ is continuous. In contrast, both the Regula-Falsi and the Secant methods assume that $f(x)$ is differentiable. The relatively high complexity of the Regula-Falsi and Secant methods compared to the bisection method leads to a reduction of function evaluations. The difference between the Regula-Falsi method and the Secant method is that in the Regula-Falsi method the domain [a b] is refined at each iteration to always contain the root, as in the bisection method. Instead, the Secant method uses always the previous root estimation at each iteration. Tab.2 also lists the computation time required for each method. The computation time depends on several factors, of which the most important is the allocation of memory. For this reason, a for loop has been added to the code so that Matlab can allocate memory in advance and get more reliable values. The calculation time values given in Tab.2 refer to the last loop of the for cycle. Tab.2 highlights that the calculation time decreases with the number of function evaluations.

Table 2: Function Evaluations and Computational Time.

Method	Function Evaluations	Computational Time
Bisection	50	3.8000×10^{-06} s
Regula-Falsi	30	2.0000×10^{-06} s
Secant	10	1.5000×10^{-06} s

Exercise 2

Let $\mathbf{f}$ be a two-dimensional vector-valued function $\mathbf{f}(\mathbf{x}) = (x_1^2 - x_1 - x_2,\, x_1^2/16 + x_2^2 - 1)^\top$, where $\mathbf{x} = (x_1, x_2)^\top$. Find the zero(s) of $\mathbf{f}$ by using Newton's method with $\partial\mathbf{f}/\partial\mathbf{x}$ 1) computed analytically, and 2) estimated through finite differences. Which version is more accurate?

This exercise is about computing the zeros of the two-dimensional function $f(x)$. A Matlab code has been implemented to compute the Jacobian analytically and to estimate the Jacobian by the forward finite difference method. Since the function $f(x)$ is a two-dimensional function, it was necessary to use the Matlab "qvector" function to represent the vector field of the function and it is exploited to set a suitable initial condition near the regions where the function could have zeros. Two different points near the two regions where the zeros could be located were chosen as initial conditions, namely $x_{01} = [3, 2]$ and $x_{02} = [-2, 1]$. The three Newton methods are based on a maximum tolerance of 10^{-8} and a maximum number of iterations for each method equal to 6. The analytical Jacobian was developed using Matlab Symbolic Toolbox, whereas FD and CD were developed using the finite difference approach. The zeros found for the analytically calculated Jacobian are given in Tab. 3. In Tab.3, the zeros are given with an accuracy of up to 4 digits to give an idea of the position of the zeros. However, if accuracy of 4 digits is also considered for the FD approach and the CD approach, the solution is practically the same.

Table 3: Estimaiton of the zeros .

Method	First zero	Second zero
Analytical Jacobian	[1.5810 0.9185]	[-0.6127 0.9881]

Therefore, in order to compare the results of the analytic Jacobian, the forward difference Jacobian, and the central difference Jacobian, it was necessary, firstly, to solve the two-dimensional function analytically with the solve fucntion provided by the Matlab symbolic toolbox and secondly, to give the normalized difference between the three Newton methods and the real solution, which is called the error. The results are given in Tab.4 and show that the analytical Jacobian and the central difference Jacobian give rise to an error smaller than the machine eps error for 6 iterations of the algorithm. Instead the solutions calculated using the newton method with the forward difference Jacobian has an error of the order of 10^{-16} for 6 iterations with respect to the second zero. This table confirms that the analitic Jacobian version of the Newton method is more accurate and, in this case, it has the same accuracy of the central difference Jacobian. On the other hand, the forward difference has instead a lower accuracy, since it is an expansion to the first order of the Jacobian.

Table 4: Errors with respect to the real zeros with 6 iterations.

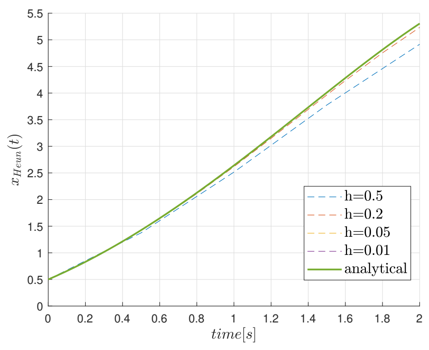
Method	Error x_1	Error x_2
Analytical Jacobian	0	0
Forward difference Jacobian	0	1.5701×10^{-16}
Central difference Jacobian	0	0

2 Numerical solution of ODE

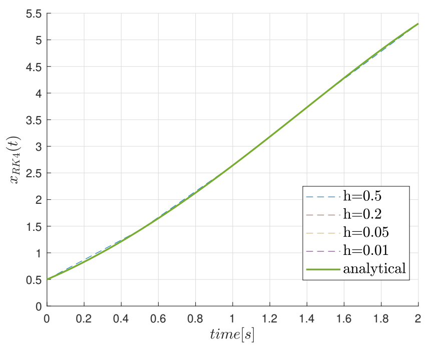
Exercise 3

The Initial Value Problem $\dot{x} = x - t^2 + 1$, $x(0) = \frac{1}{2}$, has analytic solution $x(t) = t^2 + 2t + 1 - \frac{1}{2}e^t$. 1) Implement a general-purpose, fixed-step Heun's method (RK2); 2) Solve the IVP in $t \in [0, 2]$ for $h_1 = 0.5$, $h_2 = 0.2$, $h_3 = 0.05$, $h_4 = 0.01$ and compare the numerical vs the analytical solution; 3) Repeat points 1)–2) with RK4; 4) Trade off between CPU time & integration error.

A general purpose fixed-step Heun method and an RK4 method are implemented in order to solve the initial value problem. The solutions are shown in Fig. 1. In Fig. 1a, it can be seen that the approximated solution is very close to the analytical solution when h is small. The same happens in Fig. 1b, although it is less visible since Runge Kutta 4 follows the analytic solution well even for large step sizes, since it is a 4th order method.



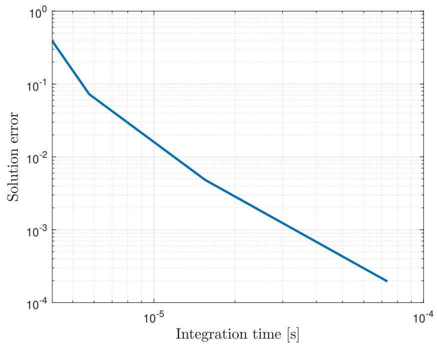
(a) Heun's Method



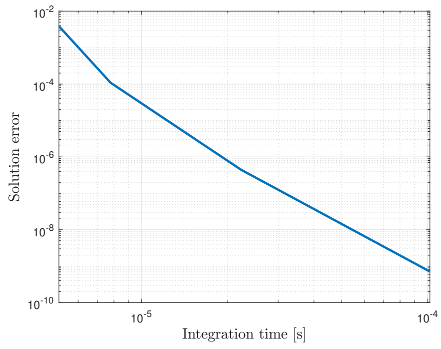
(b) RK4 Method

Figure 1: Plots of the solution using (a) Heun's method and (b) RK4 method with respect to the analytical solution.

In Figure 2 the solution errors are plotted as a function of integration time. It can be seen that the Heun method has a higher solution error but a lower integration time compared to the RK4 method. This is due to the lower function evaluation per iteration. Moreover, Figure 2a shows that when the solution error decreases, i.e., when the step h decreases, the integration time increases since there are more iterations. The same behaviour is shown in 2b, which in this case refers to the RK4 method.



(a) Heun's Method



(b) RK4 Method

Figure 2: Plots of the solution errors with respect to integration time using (a) Heun's method and (b) RK4 method.

Exercise 4

Let $\dot{\mathbf{x}} = A(\alpha)\mathbf{x}$ be a two-dimensional system with $A(\alpha) = [0, 1; -1, 2\cos\alpha]$. Notice that $A(\alpha)$ has a pair of complex conjugate eigenvalues on the unit circle; α denotes the angle from the $\text{Re}\{\lambda\}$-axis. 1) Write the operator $F_{\text{RK2}}(h, \alpha)$ that maps $\mathbf{x}_k$ into $\mathbf{x}_{k+1}$, namely $\mathbf{x}_{k+1} = F_{\text{RK2}}(h, \alpha)\,\mathbf{x}_k$. 2) With $\alpha = \pi$, solve the problem "Find $h \geq 0$ s.t. $\max\left(|\text{eig}(F(h, \alpha))|\right) = 1$". 3) Repeat point 2) for $\alpha \in [0, \pi]$ and draw the solutions in the $(h\lambda)$-plane. 4) Repeat points 1)–3) with RK4 and represent the points $\{h_i\lambda\}$ of Exercise 3 with $t = 0$. What can you say?

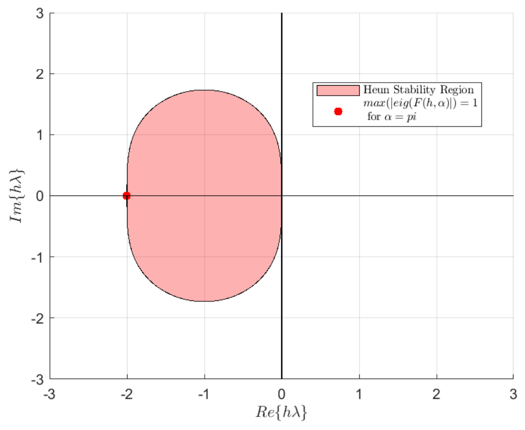
The Runge-Kutta operator of the 2nd order can be obtained using Eq.1. It is possible to implement a Matlab code that can solve the problem "Find $h \geq 0$ s.t. $\max\left(|\text{eig}(F(h, \alpha))|\right) = 1$".

$$\mathbf{F}_{\mathbf{RK2}}(h, \alpha) = \left[\mathbf{I}^{(\mathbf{n})} + \mathbf{A}(\alpha) \cdot h + \frac{(\mathbf{A}(\alpha) \cdot h)^2}{2}\right] \tag{1}$$

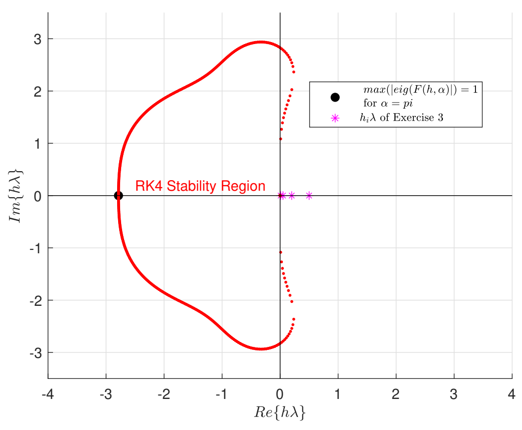
This problem is solved using the fzero function in the Matlab code. If α is fixed and equal to π the solution for the RK2 h = 2. With alpha varying from π to 0 it is necessary to add a condition related to the initial value of the Matlab fzero function. When alpha is larger than $\frac{\pi}{2}$ the initial condition of the fzero function is alpha itself. When alpha is less then $\frac{\pi}{2}$ the initial condition is set to be equal to zero. Using this Matlab code it is possible to draw the region of RK2 stability in the (hλ)-plane and the solution is shown in Fig.3a. The same procedure is repeated in the case of RK4. Firstly, the operator F_{RK4} is derived, which is defined by the eq.2.

$$\mathbf{F}_{\mathbf{RK4}}(h, \alpha) = \left[\mathbf{I}^{(\mathbf{n})} + \mathbf{A}(\alpha) \cdot h + \frac{(\mathbf{A}(\alpha) \cdot h)^2}{2} + \frac{(\mathbf{A}(\alpha) \cdot h)^3}{6} + \frac{(\mathbf{A}(\alpha) \cdot h)^4}{24}\right] \tag{2}$$

Secondly, a Matlab code is implemented to solve problem "Find $h \geq 0$ s.t. $\max\left(|\text{eig}(F(h, \alpha))|\right) = 1$". When $\alpha = \pi$ the solution obtained by the Matlab code is equal to 2.8. With alpha varying from π to 0, these conditions are developed through a trial and error approach in order to generate the typical lobes of the RK4 region of stability. The RK4 region of stabilty obtained is shown in Fig.3a. With respect to the exercise 3, A(α) is equal to 1. Therefore the eigenvalue of the 1x1 A matrix is equal to 1. The position of the $h_i\lambda$ points is highlighted in Fig3b. If the Rk4 method is applied to the exercise 3 IVP, the numerical solution will not be the correct one since all the points stand outside the RK4 stability region.



(a) Heun Method stability region



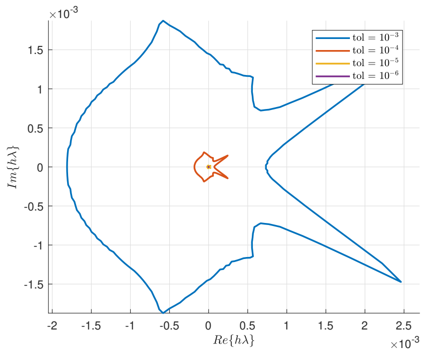
(b) RK4 stability Region

Figure 3: Plots of the stability regions of Heun method and RK4 method in $(h\lambda)$-plane.

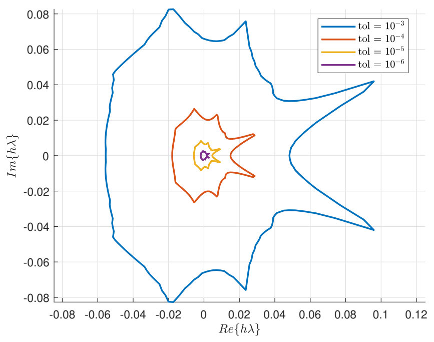
Exercise 5

Consider the IVP $\dot{\mathbf{x}} = A(\alpha)\mathbf{x}$, $\mathbf{x}(0) = [1, 1]^T$, to be integrated in $t \in [0, 1]$. 1) Take $\alpha \in [0, 2\pi]$ and solve the problem "Find $h \geq 0$ s.t. $\|\mathbf{x}_{\text{an}}(1) - \mathbf{x}_{\text{RK1}}(1)\|_\infty = \text{tol}$", where $\mathbf{x}_{\text{an}}(1)$ and $\mathbf{x}_{\text{RK1}}(1)$ are the analytical and the numerical solution (with RK1) at the final time, respectively, and $\text{tol} = \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. 2) Plot the five locus of solutions in the $(h\lambda)$-plane; plot also the function evaluations vs tol for $\alpha = \pi$. 3) Repeat points 1)–2) for RK2 and RK4.

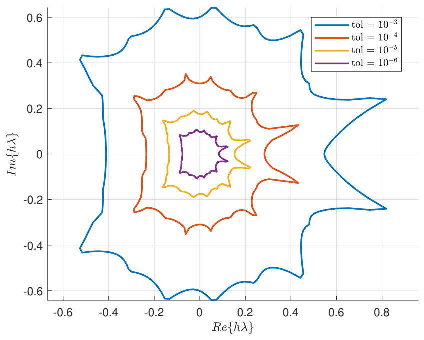
A Matlab code is implemented to solve the problem "Find $h \geq 0$ s.t. $\|\mathbf{x}_{\text{an}}(1) - \mathbf{x}_{\text{RK1}}(1)\|_\infty = \text{tol}$". In the Matlab code it is used the fzero function inside a for cycle to solve the problem. The initial condition used in the fzero function is equal to the tolerance used in the problem. However, a further condition is added. When the solution of the problem is NaN the fzero initial condition is set to be equal to the previous value of h evaluated inside the iteration. All these conditions comes out from a trial and error approach based on the minimization of the computational time required to execute the code and in order to avoid singularities during the execution. Another important aspect of the code to highlight is that the fzero function vary the time step h in order to find the solution of the problem "Find $h \geq 0$ s.t. $\|\mathbf{x}_{\text{an}}(1) - \mathbf{x}_{\text{RK1}}(1)\|_\infty = \text{tol}$". For this reason it is necessary to implement a condition into the RK1 code representing the Matlab formulation of the RK1 method. The condition is related to the fact that the variation of the step value inside a finite range of time may generate a non integer number of steps. With this adjustment the code overcomes the problem generated by a non integer number of steps. The solution obtained for the RK1 method is shown in figure 4a. The same procedure is done for RK2 and RK4 methods, with the distinction that the RK1 one requires more computational time as compared to RK2 and RK4 and the results are shown in Fig.4b and Fig.4c.



(a) Solution of the problem using RK1



(b) Solution of the problem using RK2



(c) Solution of the problem using RK4

Figure 4: Plots of the locus of solution of the Problem for the three Runge-kutta algorithms.

Finally, the relation among the three Runge-Kutta methods, the function evaluations and the tolerances fixing $\alpha = \pi$ are are plotted in figure 5. RK1 requires the highest function evaluations compared to RK2 and RK4, since it is a method of the first order. The three algorithms show the same pattern as the tolerance increases. In fact, the function evaluations decrease when the tolerance is increased. RK4 has the lowest function evaluations, which is due to the higher order of the algorithm compared to the others.

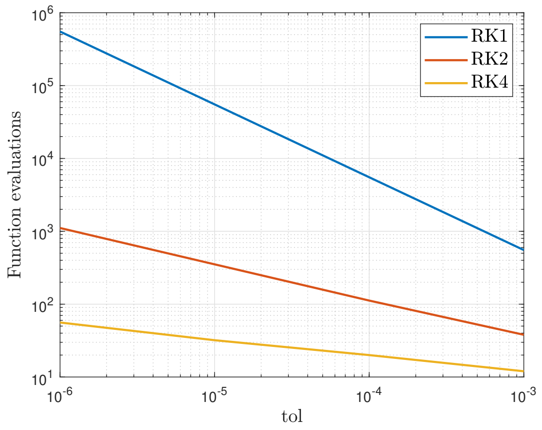


Figure 5: Function evaluations vs tolerance for $\alpha = \pi$

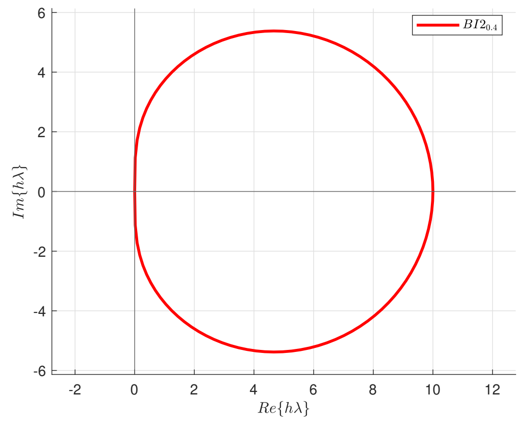
Exercise 6

Consider the backinterpolation method BI$2_{0.4}$. 1) Derive the expression of the linear operator $B_{\mathrm{BI2}_{0.4}}(h,\alpha)$ such that $\mathbf{x}_{k+1} = B_{\mathrm{BI2}_{0.4}}(h,\alpha)\mathbf{x}_k$. 2) Following the approach of point 3) in Exercise 5, draw the stability domain of BI$2_{0.4}$ in the $(h\lambda)$-plane. 3) Derive the domain of numerical stability of BI2_θ for the values of $\theta = [0.1, 0.3, 0.7, 0.9]$.

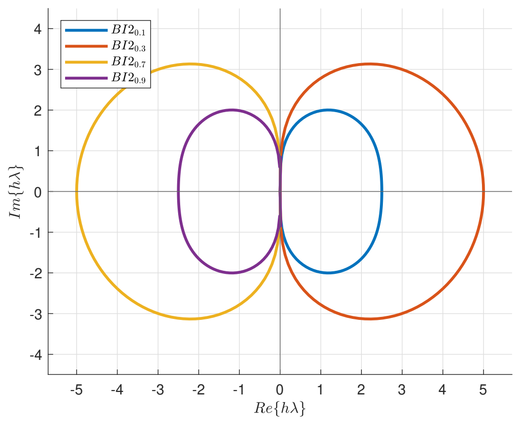
The derivation of the operator $B_{BI2_{0.4}}$ of the backinterpolation method is obtained by substituting $\vartheta = 0.4$ into eq.3.

$$\mathbf{B}_{BI2\vartheta} = \left[\mathbf{I^{(n)}} - \mathbf{A}(\alpha)(1-\vartheta)h + \frac{(\mathbf{A}(\alpha)(1-\vartheta)h)^2}{2!}\right]^{-1} \left[\mathbf{I^{(n)}} + \mathbf{A}(\alpha)\vartheta h + \frac{(\mathbf{A}(\alpha)\vartheta h)^2}{2!}\right] \quad (3)$$

Once the $B_{BI2_{0.4}}$ operator is obtained, it is possible to set up a Matlab code to retrieve the stability domain in $(h_i\lambda)$-plane by varying α from π to 0. To find the stability domain it is necessary to solve the problem "Find $h \geq 0$ s.t.max $(|\mathrm{eig}(B(h,\alpha))|) = 1$". In the Matlab code it is used the fzero function to solve the problem. During the research of the problem possible solution h the operator $\mathbf{B}_{BI2_{0.2}}$ might generate a singular matrix. A try/catch statement is added in the code in order to make the code overcome the problem and work even when the first part of the $\mathbf{B}_{BI2_{0.2}}$ is singular. As initial condition of the fzero function is used $h0 = 9$, this number comes out from a trial and error approach based on the minimization of the computational time required by Matlab to execute the code. The stability domain is plotted in Fig.6a and the stability region is outside the pseudo-circumference. In the second part of the exercise the general $\mathbf{B}_{BI2_\theta}$ operator displayed in Eq.3 is used with four different values of θ and they are [0.1, 0.3, 0.7, 0.9]. The previous Matlab code is improved to evaluate the stability region with different θ. The solution obtained is shown in Fig.6b. It can be seen that as θ increases,the pseudo-circumference size decreases. For θ larger than $\frac{\pi}{2}$, the pseudo-circumference is on the left hand side of the complex plane, but when it is smaller than $\frac{\pi}{2}$, the pseudo-circumference is on the right hand side of the complex plane.



(a) $BI2_{0.4}$ region of stability.



(b) Region of stability of $BI2_{0.1}$ $BI2_{0.3}$ $BI2_{0.7}$ $BI2_{0.9}$.

Figure 6: Plots of the region of stability for different $BI2_\theta$ methods.

Exercise 7

Consider the IVP $\dot{\mathbf{x}} = B\mathbf{x}$ with $B = [-180.5, 219.5; 179.5, -220.5]$ and $\mathbf{x}(0) = [1, 1]^T$ to be integrated in $t \in [0, 5]$. Notice that $\mathbf{x}(t) = e^{Bt}\mathbf{x}(0)$. 1) Solve the IVP using RK4 with $h = 0.1$; 2) Repeat point 1) using BI2$_{0.1}$; 3) Compare the numerical results in points 1) and 2) against the analytic solution; 4) Compute the eigenvalues associated to the IVP and represent them on the $(h\lambda)$-plane both for RK4 and BI2$_{0.1}$; 5) Discuss the results.

The initial value problem is first solved using the RK4 method with a step h = 0.1. A Matlab code is implemented using the RK4 operator defined in 2. However, since this is a stiff problem, the RK4 algorithm cannot find a solution. In fact, as shown in Fig.7 the analytical solution and the numerical solution diverge. If instead a backinterpolation method is used, the problem can be solved. In this report the $BI2_{0.1}$ algorithm is used. It is implemented a Matlab code based on the $BI2_{0.1}$ operator, which can be retrieved from the general $BI2_\vartheta$ operator defined in eq.3 substituting $\vartheta = 0.1$. Fig.7 shows the solutions of the two components of the x-vector compared to the analytical solution, they are practically overlapping. Using RK4 algorithm instead, the numerical solution diverge from the analytic solution because RK4 cannot handle stiff problems. The eigenvalues associated with the B matrix are calculated and

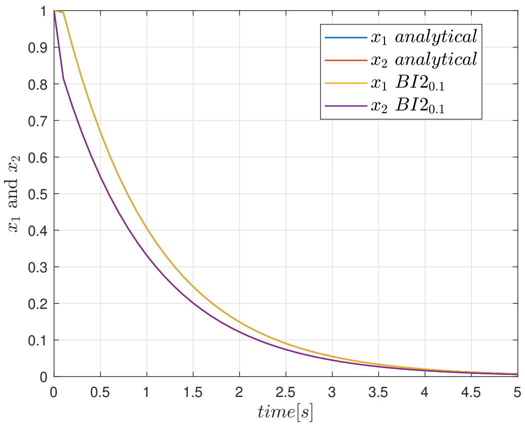


Figure 7: $BI2_{0.1}$ results vs analytical results.

multiplied with the step h. Doing so it is possible to represent the eigenvalues associated to the IVP and represent them on the (hλ)-plane. The two eigenvalues are shown in Fig.8. One is equal to -400 and the second one is equal to -1. In order to compare the position of the eigenvalues in the (hλ)-plane, part of the code used to plot the region of stability of the RK4 and $BI2_{0.1}$ in exercise 4 is used in this exercise. Fig.8 shows the two eigenvalues and the region of stability of the two algorithms in the (hλ)-plane. Only one of the two eigenvalues is inside the RK4 region of stability. However, both eigenvalues are inside the $BI2_{0.1}$ region of stability, which is the area outside the pseudo-circumference, and for this reason it is possible to find a numerical solution which follow the trend of the analytical solution.

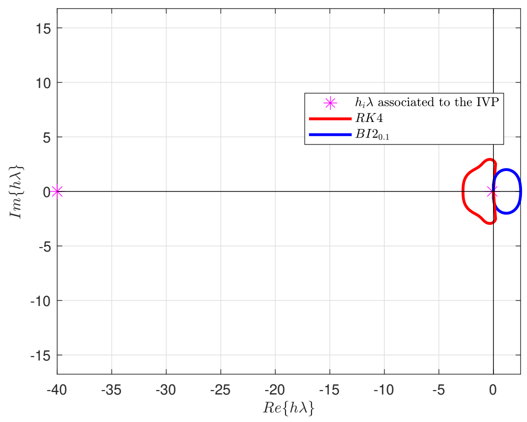


Figure 8: Representation of the eigenvalues of the IVP problem on the $(h\lambda)$-plane with RK4 and $BI2_{0.1}$ regions of stability.