MSAS – Assignment #1: Simulation

Davide Iafrate, 962657

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.

To guess an interval in which to search for the zero the function is plotted. A suitable interval in which to begin the search for the zero-crossing is identified as $[a, b] = [0.5; 1.1]$
The bisection method relies on restricting the search interval [a,b] at each iteration, by checking

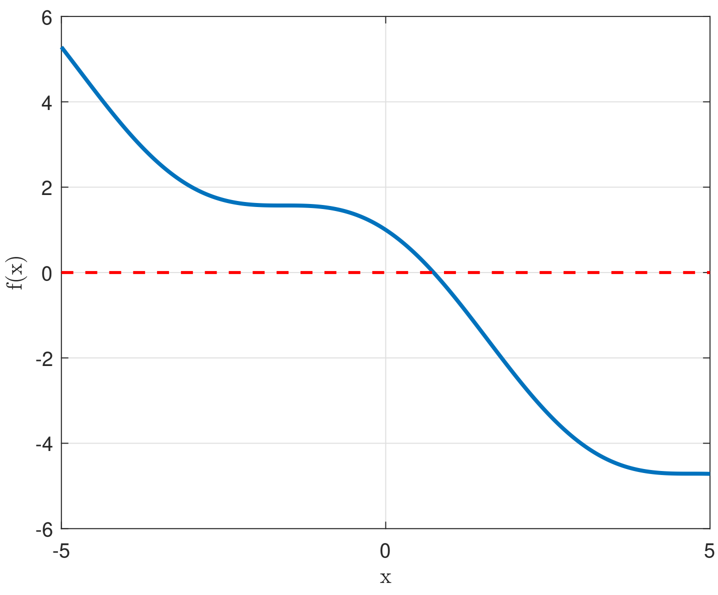


Figure 1: Function plot

the sign of the function evaluated at its extremes

$$x = \frac{a+b}{2} \tag{1}$$

$$\begin{cases} if f(a)f(x) < 0 \\ \quad b = x \\ else \\ \quad a = x \end{cases} \tag{2}$$

The secant method instead operates by iterating on the equation:

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \tag{3}$$

while the regula falsi method is a combination of the two.

The methods are stopped when the search interval is narrowed down to below the required tolerance. The faster method turns out to be the Secant, requiring the least number of function evalutaions as seen in Tab.[1].

Table 1: Root-finding methods comparison

Method	Solution	Function Evaluations	Computational time [s]
Bisection	0.739085132256150	30	3.592e-06
Secant	0.739085133215161	7	1.092e-06
Regula Falsi	0.739085133215161	9	1.192e-06

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?

In order to have an initial guess for Newton's algorithm, the function is plotted using MATLAB's "quiver" function Fig[2], showing two plausible initial guesses near which to search for the zeros ([x_1,x_2]=[1.5; 1] and [-1;1]).

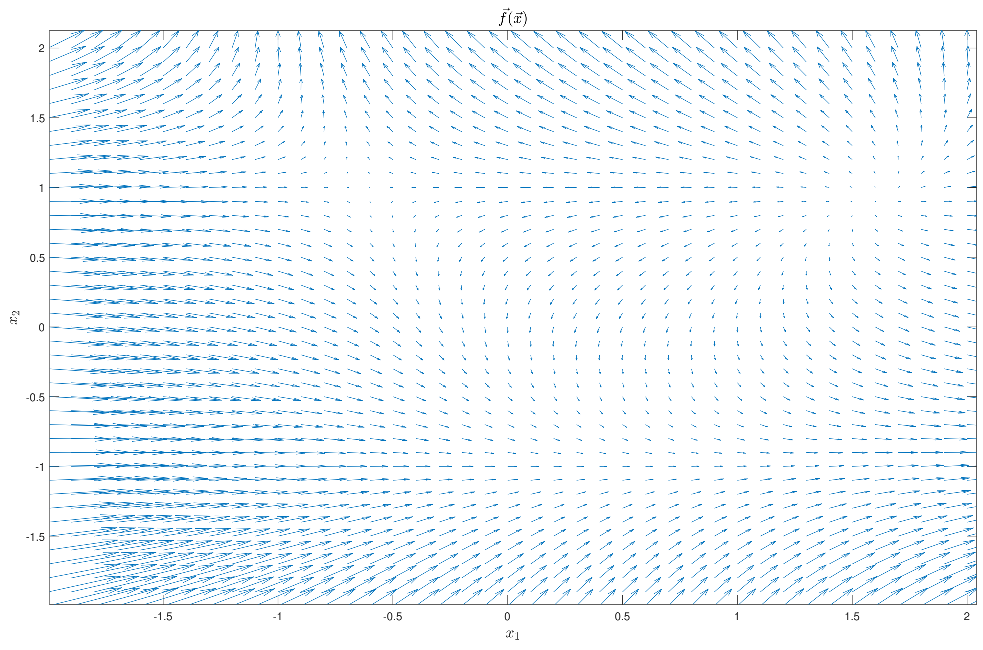


Figure 2: quiver plot of $\mathbf{f}(\mathbf{x})$

Newton's method is employed to compute the zeros iteratively by solving at each k step the equation:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}(\mathbf{x}_k)^{-1}\mathbf{f}(\mathbf{x}_k) \tag{4}$$

with the Jacobian being calculated both analytically

$$\mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \tag{5}$$

and with the forward and central *finite difference* approximations. In these methods each column of the Jacobian $\mathbf{J}$ is calculated by perturbing the function in each i-th direction $\mathbf{e}_i$ by a quantity ϵ_i:

$$\left.\frac{\partial \mathbf{f}}{\partial x_i}\right|_{forward} \approx \frac{\mathbf{f}(\mathbf{x}+\mathbf{e}_i\epsilon_i)-\mathbf{f}(\mathbf{x})}{\epsilon_i} \tag{6}$$

$$\left.\frac{\partial \mathbf{f}}{\partial x_i}\right|_{central} \approx \frac{\mathbf{f}(\mathbf{x}+\mathbf{e}_i\epsilon_i)-\mathbf{f}(\mathbf{x}-\mathbf{e_i}\epsilon_\mathbf{i})}{2\epsilon_i} \tag{7}$$

$$\tag{8}$$

each perturbation size ϵ_i being computed using the the machine floating-point accuracy **eps** as

$$\epsilon_i = \begin{cases} |x_i|\sqrt{eps} & \text{if } |x_i| \geq 1 \\ \sqrt{eps} & \text{if } |x_i| < 1 \end{cases} \tag{9}$$

and implemented in MATLAB as `epsilon_i = max(sqrt(eps), sqrt(eps)*abs(x(i)))`. The stopping criterion is selected as a maximum number of iterations Nmax, given as argument to the function (in this case Nmax $= 5$).

Denoting with $\overline{\mathbf{x}}^{(i)}$, $(i = 1, 2)$ the analytical solution for each of the two zeros we can compare the solutions found by using Newton's algorithm with both the analytical jacobian and the numerical one, computed with both *forward* and *central* finite difference schemes by analyzing the Euclidean norm of their difference Tab.[2]:

Table 2: Analytical and finite difference (FD) Jacobian comparison

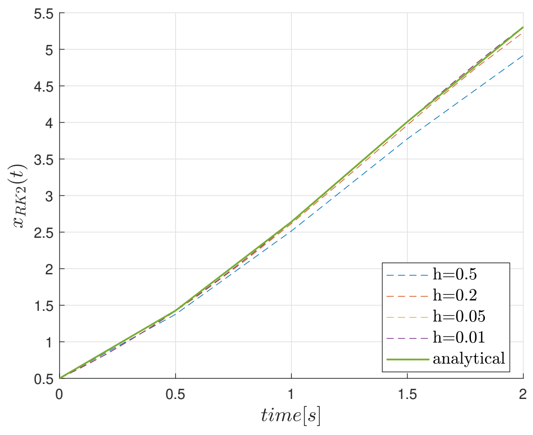
Jacobian algorithm	$\|\|\mathbf{x}^{(1)} - \overline{\mathbf{x}}^{(1)}\|\|_2$	$\|\|\mathbf{x}^{(2)} - \overline{\mathbf{x}}^{(2)}\|\|_2$
Analytical	0	1.11e-16
FD forward	2.2204e-16	0
FD central	1.11e-16	0

2 Numerical solution of ODE

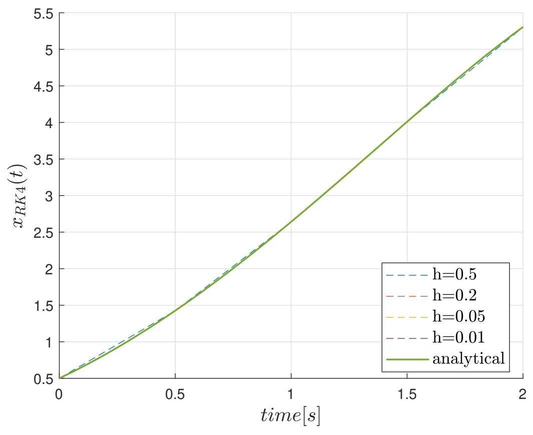
Exercise 3

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.

We can see that in the RK2 case the solutions follow closely the analytical one only in the cases of small timesteps Fig.[3a], while for the RK4 integrator even the larger timesteps produce an accurate result Fig.[3b].



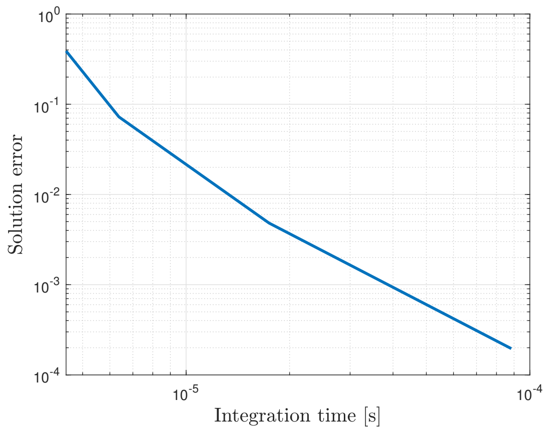
(a) RK2 integration



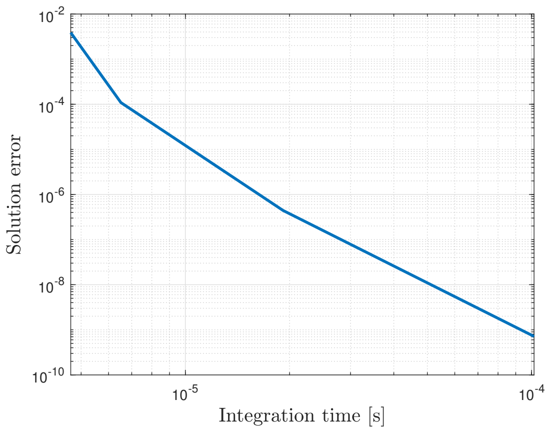
(b) RK4 integration

Figure 3: RK2 - RK4 integrators comparison

The integrators have been designed to return a vector containing the state value at each integrated timestep, so the error of the RK4 and RK2 methods has been computed as $max(|x_{RK} - x_{analytical}|)$ and we can see that in the case of RK4 we can achieve much smaller solution errors for the same integration time Fig.[4]



(a) RK2 tradeoff



(b) RK4 tradeoff

Figure 4: RK2 - RK4 integration times/accuracy tradeoff

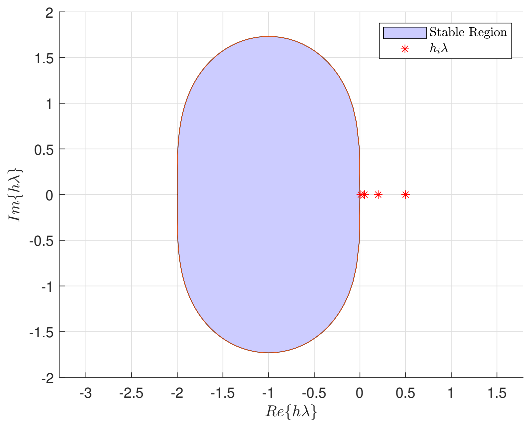
Exercise 4

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 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 4 with $t = 0$. What can you say?

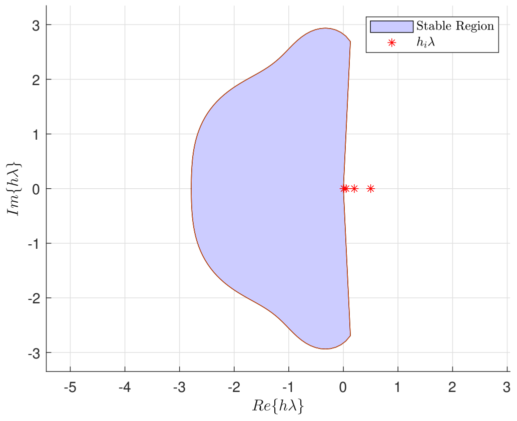
The operators for the two Runge-Kutta methods are simply the Taylor expansion of the dynamics truncated at the required order:

$$\mathbf{F}_{RK2}(h,\alpha) = \mathbf{I} + h\mathbf{A} + \frac{(h\mathbf{A})^2}{2!}$$
$$\mathbf{F}_{RK4}(h,\alpha) = \mathbf{I} + h\mathbf{A} + \frac{(h\mathbf{A})^2}{2!} + \frac{(h\mathbf{A})^3}{3!} + \frac{(h\mathbf{A})^4}{4!}$$

By solving the assigned problem we find the region of marginal stability of each method Fig.[5]. Plotting the mapped eigenvalue of the dynamical system $\dot{x} = x - t^2 + 1$ for t=0 (i.e. $\lambda = 1$) in the four cases of $h = [0.5; 0.2; 0.05; 0.01]$ we see that they are correctly mapped in the unstable region of both methods, thus resulting in a (correctly) unstable dynamics for t=0.



(a) Runge-Kutta 2



(b) Runge-Kutta 4

Figure 5: Stability regions

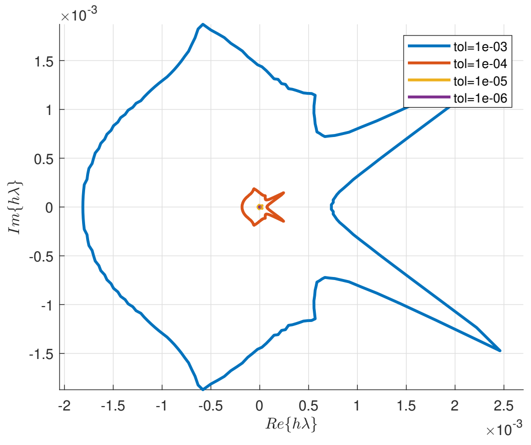
Exercise 5

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 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.

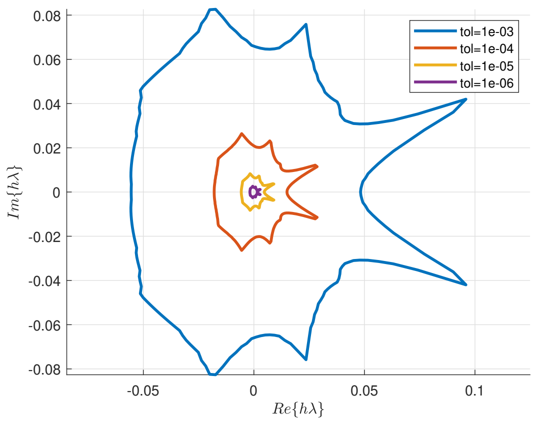
In this problem the analytical solution is computed as:

$$\mathbf{x}_{an}(t) = e^{\mathbf{A}(\alpha)t}\mathbf{x}_0 \tag{10}$$

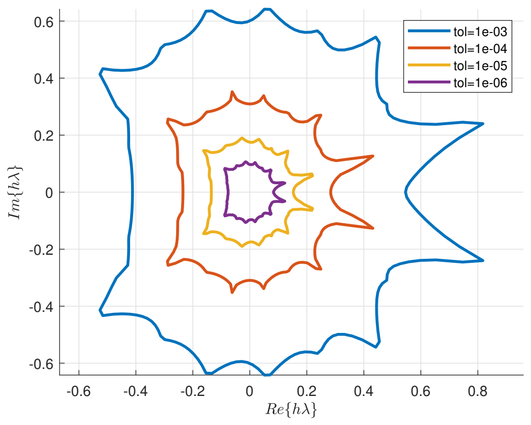
By solving the given problem using MATLAB's `fzero` function, the stepsize resulting in a given tolerance on each state component is found. It can be seen that, for the prescribed tolerances, RK4 requires a stepsize in the order of 10^{-1}, RK2 requires a stepsize on the order of 10^{-2} while RK1 requires stepsizes in the order of each tolerance, all the way down to 10^{-6}, thus requiring many more steps to be taken in order to complete the integration.



(a) RK1



(b) RK2



(c) RK4

Figure 6: Accuracy domains of Runge-Kutta schemes

For each method the function evaluations necessary to compute the solution up to a determined tolerance with respect to the exact one are computed and plotted against each other Fig[7], again pointing out the much higher efficiency of RK4.

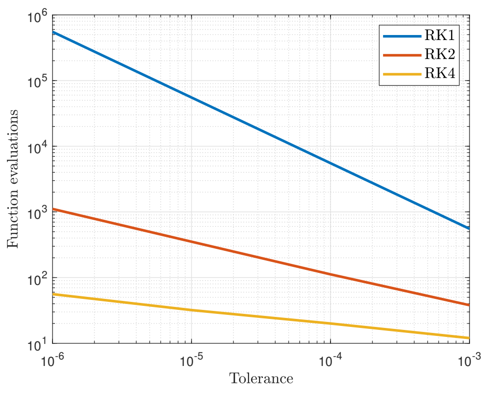


Figure 7: Function evaluations vs solution tolerance for $\alpha = \pi$

The time required to plot the RK1 accuracy domain is significantly higher than the respective time for RK4 and RK2, due to the much higher number of steps required, especially with the tightest tolerances. On an AMD Ryzen 5 4500H CPU with 16GB of RAM computing all three plots requires about 2 minutes.

Exercise 6

Consider the backinterpolation method $\mathrm{BI2}_{0.4}$. 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 $\mathrm{BI2}_{0.4}$ in the $(h\lambda)$-plane. 3) Derive the domain of numerical stability of $\mathrm{BI2}_\theta$ for the values of $\theta = [0.1, 0.3, 0.7, 0.9]$.

The backinterpolation method $BI2_\theta$ is made up of a RK2 forward step of length θh and a *backward* RK2 step of length $(1-\theta)(-h)$.

The resulting linear operator is:

$$B_{BI2_\theta} = [I - Ah(1-\theta) + \frac{(Ah)^2}{2}(1-\theta)^2]^{-1}[I + Ah\theta + \frac{(Ah)^2}{2}\theta^2] \tag{11}$$

The marginal stability region approaches the imaginary axis as θ increases as seen in Fig[9]. In the case of $\theta < 0.5$ the method is stable **outside** of the marginal stability line, while in the case of $\theta > 0.5$ the method is stable **inside** the marginal stability line. This can be seen by plotting the contour lines for which the absolute value of the mapped eigenvalues is 0.9 and 1.1 (respectively stable and unstable) in the $h\lambda$ plane Fig.[9].

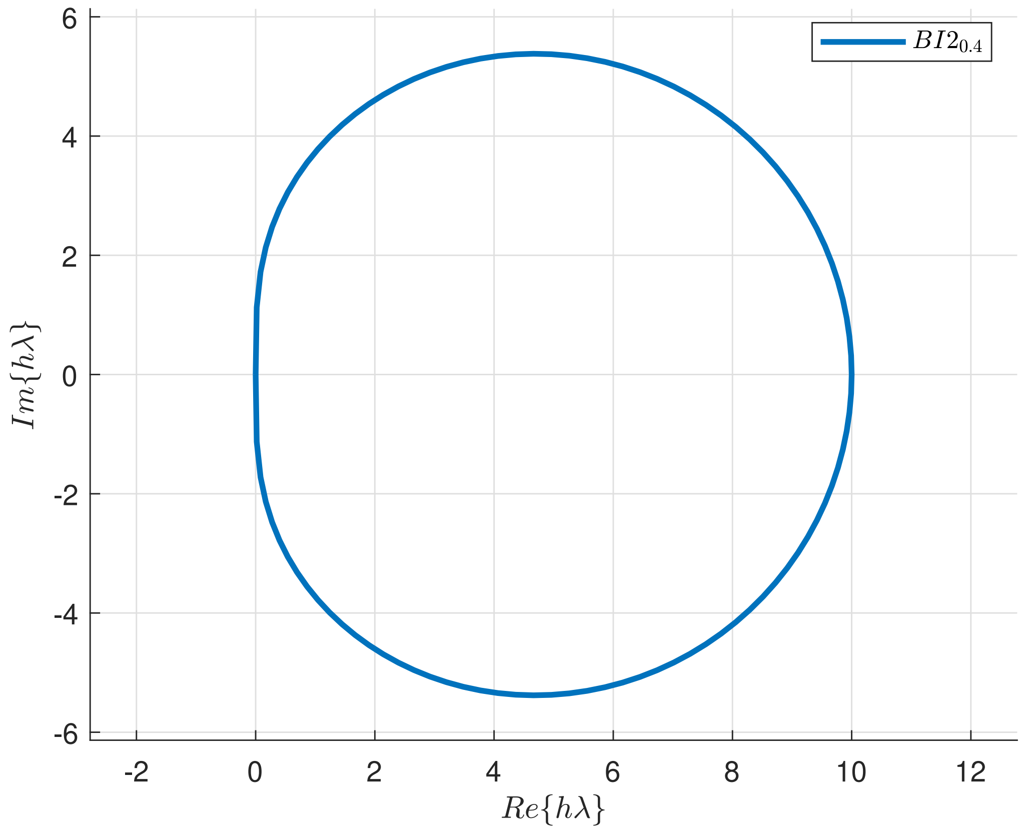


Figure 8: $BI_{0.4}$ stability regions

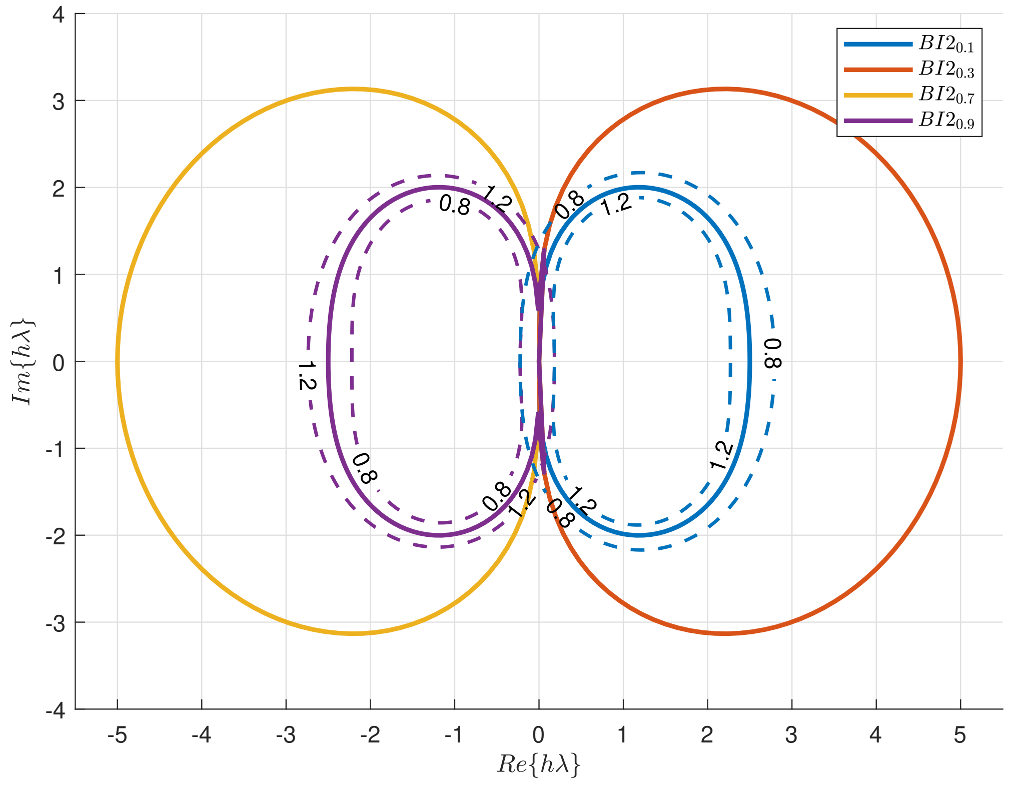


Figure 9: BI_θ stability regions for different θs

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.

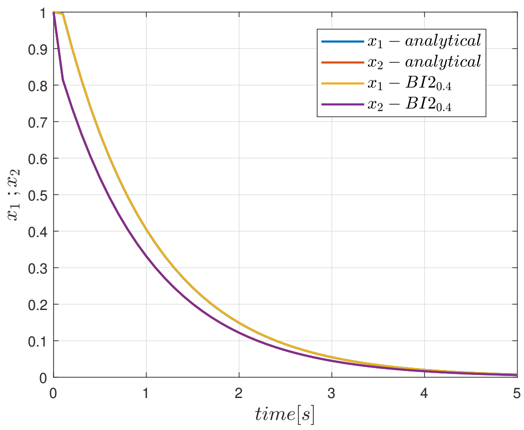
The given dynamical system is stiff, as it has two eigenvalues which have different orders of magnitude.

Using the RK4 integrator and the $BI2_{0.1}$ integrators we can compute numerically the trajectory of the state $\mathbf{x}(t)$, using a stepsize $h = 0.1$. With this stepsize we can see that the $BI2_{0.1}$ integration (Fig.[10a]) follows very accurately the numerical solution while the $RK4$ integration(Fig.[10b]) does not, as the state "explodes" to huge values.

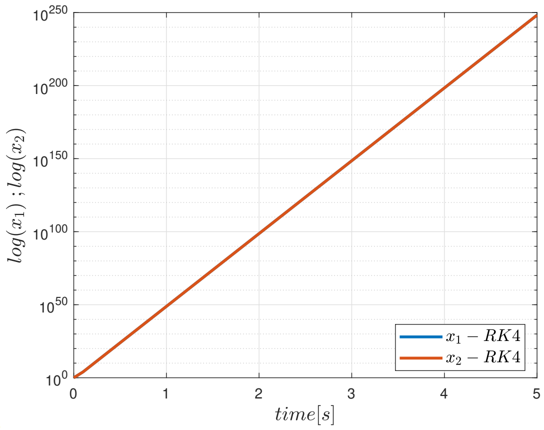
This is motivated by the fact that one of the eigenvalues of the system is mapped outside of the RK4 stability region (i.e. **inside** of the black line) for the given stepsize h, while in the $BI2_{0.1}$ integration *both* eigenvalues are inside the stability region (**outside** the turquoise line)

In this case the correct choice for integration would be $BI2_{0.1}$.

2.1 Bibliography

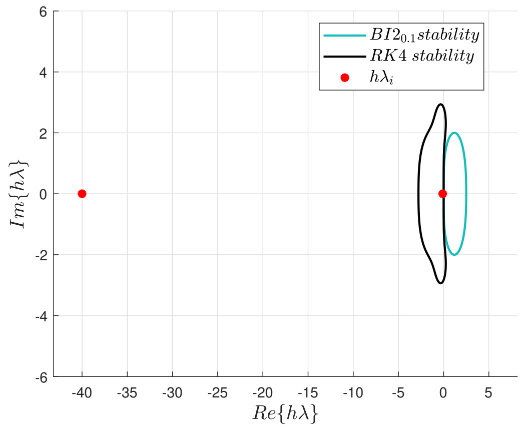


(a) $BI2_{0.1}$ integrator vs analytical

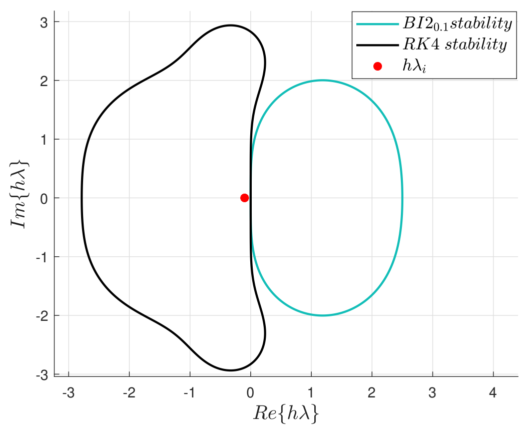


(b) RK4 integration

Figure 10: RK4-$BI2_{0.1}$ comparison



(a) Location of both eigenvalues



(b) Closeup of the smallest eigenvalue location

Figure 11: Eigenvalues location in the methods' stability regions