Matlab buit-in solvers `bvp4c` and `bvp5c`

Chiara Garuglieri, Valentina Sanson, Claudia Scarpa

November 2025

1 Introduction

In this project, we explore boundary value problems (BVPs), starting with their theoretical foundations and then examining their practical applications. We will begin by reviewing the key concepts behind these problems, providing the necessary theoretical background. Boundary value problems arise in many areas of applied mathematics, physics and engineering, where the behavior of a system is constrained by conditions imposed at more than one point of the domain. Unlike initial value problems (IVPs), which typically admit a unique solution under appropriate assumptions, BVPs may have a unique solution, multiple solutions or none at all. For this reason, finding a solution is challenging, and robust and flexible computational methods are required.

In this project we focus on the theoretical foundations and the numerical solutions of BVPs, with particular attention to collocation-based techniques. After presenting the mathematical background, we will investigate two built-in MATLAB solvers based on Lobatto IIIa formulas of different orders - `bvp4c` and `bvp5c` - to understand how they work, compare their main differences and identify the scenarios in which each solver is most appropriate. Although the two solvers share the same underlying methodology, they differ in accuracy, stability, residual control and computational efficiency. Understanding these differences is crucial for selecting the most appropriate solver in practical applications.

To supplement the theoretical analysis, we developed a graphical user interface (GUI) in MATLAB App Designer that enables the user to experiment interactively with several examples of BVPs. The examples implemented in the app are derived from the classical paper by Shampine, Reichelt, and Kierzenka (2000), *Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c* (2). This collection of test problems is a standard reference in the literature and provides a rigorous and diverse set of challenges for evaluating BVP solvers.

The final part of the project presents an analysis of selected case studies, highlighting the strengths and limitations of each solver and offering practical guidelines for their use in different scenarios.

2 Theoretical Background

2.1 Boundary Value Problems

Ordinary differential equations (ODEs) model phenomena that evolve continuously in time. A system of ODEs alone generally admits infinitely many solutions; additional conditions must be imposed to characterize a unique physical solution. When these constraints are prescribed as the values of all components at a single point of the domain, one obtains an initial value problem (IVP). In many applications, however, the behavior of the solution must satisfy conditions imposed at two or more distinct points of the domain. This leads to a boundary value problem (BVP) of the form

$$\begin{aligned} y' &= f(t,y), \qquad a < t < b \\ g(y(a),y(b)) &= 0 \end{aligned} \tag{1}$$

where $f : \mathbb{R}^{n+1} \to \mathbb{R}^n$, and the boundary operator $g : \mathbb{R}^{2n} \to \mathbb{R}^n$ encodes the conditions.

A BVP may admit multiple solutions, no solution at all, or may require the identification of unknown parameters to satisfy the boundary conditions. For this reason, most numerical approaches require an initial guess for both the solution and any unknown parameters.

A classical viewpoint relates a BVP to a parameter-dependent IVP. Introducing a vector $s \in \mathbb{R}^n$, consider the initial value problem

$$\begin{aligned} w' &= f(t,w), \\ w(a) &= s, \quad t > a. \end{aligned} \tag{2}$$

For each s, the IVP admits a unique solution $w(t;s)$ under standard existence and uniqueness assumptions. For a vector s^*, w solves the BVP when the boundary condition is satisfied:

$$\phi(s^*) \equiv g(s^*, w(b;s^*)) = 0. \tag{3}$$

Thus, solving the BVP amounts to finding roots of $\phi(s)$. This establishes a clear connection between nonlinear algebraic equations and BVPs. The following theorem relates the number of roots to the number of BVP solutions:

Theorem 1 *Suppose that $f(t,y)$ is continuous on $D = \{(t,y) : a \le t \le b, \|y\| < \infty\}$ and satisfies a uniform Lipschitz condition in y. Then the BVP* (1) *has many solutions as there are distinct roots s^* of* (3)*. For each s^* satisfying $\phi(s^*) = 0$ a solution of the BVP is given by:*

$$y(\cdot) = w(\cdot; s^*).$$

2.2 Collocation Method as a Weighted Residual Method

Among the numerical techniques for BVPs - shooting, finite differences and weighted residual methods - the collocation approach plays a central role in MATLAB's solvers `bvp4c` and `bvp5c`. Collocation belongs to the family of weighted residual methods, but it adopts a particularly simple choice of test functions, namely Dirac delta distributions $\delta(x - x_j)$, where x_j are the collocation nodes. In the collocation method, the approximate solution is therefore obtained by enforcing that the differential equation is satisfied exactly at these nodes.

To illustrate the idea, consider a linear BVP with Dirichlet conditions:

$$\begin{aligned} -(k(x)u'(x))' + q(x)u(x) &= f(x), \quad 0 < x < 1 \\ u(0) = u(1) &= 0 \end{aligned}$$

The problem can be rewritten using the linear functional $\mathscr{L}[u] = -(k(x)u'(x))' + q(x)u(x)$ as

$$\mathscr{L}[u] = f(x), \quad u(0) = u(1) = 0, \quad 0 < x < 1.$$

It may also be reformulated as a variational problem by introducing a test function v. In fact multiplying the equation by a test function $v \in L^2[0,1]$ and integrating over the domain, we get

$$\int_0^1 v(x)(-(k(x)u'(x))' + q(x)u(x) - f(x))\, dx \;=\; (v,\, \mathscr{L}[u] - f) \;=\; 0.$$

This leads to the variational formulation of the BVP:

$$\text{find } u \quad \text{s. t.} \quad (v,\, \mathscr{L}[u] - f) \;=\; 0, \quad \forall v \in L^2(0,1).$$

To obtain an approximate solution, we choose finite-dimensional subspaces $S^N \subset C^2(0,1)$ (trial space) and $\tilde{S}^N \subset L^2(0,1)$ (test space) with bases $\{\phi_i\}_{i=1}^N$ and $\{\psi_j\}_{j=1}^N$. We define

$$u(x) \approx U(x) = \sum_{j=1}^N c_j \phi_j(x),$$
$$v(x) \approx V(x) = \sum_{j=1}^N d_j \psi_j(x).$$

Therefore substituting V and U and using the linearity of inner product we get:

$$(V,\, \mathscr{L}[U] - f) \;=\; \sum_{j=1}^N d_j(\psi_j,\, \mathscr{L}[U] - f) \;=\; 0 \quad \forall V \quad \iff \quad (\psi_j,\, \mathscr{L}[U] - f) = 0 \quad \forall j.$$

$$\iff \quad \sum_{i=1}^N \left(\int_0^1 \psi_j(x)(-k(x)\phi_i'(x))'dx + \int_0^1 \psi_j(x)q(x)\phi_i(x)dx \right) c_i = \int_0^1 \psi_j(x)f(x)dx$$

So the goal now is to find the coefficients c_i by solving a linear system of N equations $Ac = F$ where:

$$A_{j,i} = \int_0^1 \psi_j(x) \left[(-k(x)\phi_i'(x))' + q(x)\phi_i(x) \right] dx \quad \text{and}$$
$$F_j = \int_0^1 \psi_j(x)f(x)dx.$$

2.3 The Collocation Method

Different choices of test and trial functions lead to different projection methods. In the collocation approach, the test functions are chosen as Dirac delta distributions:

$$\psi_j(x) = \delta(x - x_j), \quad 0 < x_0 < x_1 < \; ... \; < x_N < 1.$$

This choice simplify a lot the construction of the linear system, since inner product with Dirac delta functions becomes essentially a function evaluation at the point x_j. Indeed we have:

$$A_{j,i} = (-k(x_j)\phi_i'(x_j))' + q(x_j)\phi_i(x_j), \qquad j,i = 1,\, ...,\, N$$
$$F_j = f(x_j), \qquad j = 1,\, ...,\, N.$$

In this way we obtain the collocation method, which constructs the approximated solution U just imposing that the differential equation is satisfied at some nodes. This yields a linear algebraic system whose solution determines the coefficients of the approximate solution U. Collocation methods provide high-order accuracy and produce piecewise polynomial approximations that are smooth on the entire domain. Both `bvp4c` and `bvp5c` implement collocation formulas based on Lobatto IIIa Runge–Kutta schemes.

2.3.1 Lobatto IIIa Collocation Formulas

The Lobatto IIIa formula is an implicit Runge-Kutta method based on collocation at the endpoints of the integration interval. It is symmetric, A-stable and for this reason it is often used for stiff differential equations. In general, for a step size h, the method computes internal stages Y_i by solving an implicit system involving the derivative function $f(t,y)$ and then updates the solution y_{n+1} as a weighted combination of these stages.

For example, the 3-stage Lobatto IIIa method, that is the one used in `bvp4c`, has the following Butcher tableau:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{5}{24} & \frac{1}{3} & -\frac{1}{24} \\ 1 & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \hline & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{array}$$

The stages Y_1, Y_2, Y_3 are found by solving the implicit equations and the next step y_{n+1} is obtained as:

$$y_{n+1} = y_n + h\sum_{i=1}^{3} b_i f(t_n + c_i h, Y_i)$$

This 3-stage method achieves 4th order accuracy while preserving stability.

3 MATLAB Solvers

3.1 The bvp4c solver

The MATLAB solver `bvp4c` is a widely used tool for the numerical solution of boundary value problems for ordinary differential equations. Its formulation is based on a collocation approach that employs the three-stage Lobatto IIIa method. The implementation described by Shampine and collaborators includes an additional correction that yields an overall accuracy of fourth-order. This enhancement, combined with the smoothness properties of the collocation polynomials, makes `bvp4c` a robust and effective solver for a broad class of linear and nonlinear BVPs.

The problem must first be rewritten as a system of first-order equations of the form

$$y' = f(x,y,p), \quad a \le x \le b,$$

where p denotes any unknown parameters that must be determined so that the boundary conditions

$$g(y(a), y(b), p) = 0,$$

are satisfied. Once this formulation is provided, the solver constructs a numerical approximation of the solution in the form of a piecewise polynomial function $S(x)$. In each subinterval of the mesh, S is a polynomial of degree three or higher, determined so that the differential equation is satisfied exactly at the Lobatto IIIa collocation nodes, which include the endpoints and the midpoint of each interval. These conditions produce a nonlinear algebraic system solved using Newton's method.

One central feature of this construction is that the collocation polynomials are required to match not only in value but also in derivative at the subinterval endpoints. As a consequence, the final approximation satisfies $S \in C^1[a,b]$. This global smoothness is essential for defining the residual of the approximation,

$$r(x) = S'(x) - f(x, S(x))$$

which measures how closely a collocation polynomial satisfies the differential equation. Since the residual is well-defined everywhere, even for sparse meshes, it provides an effective way to detect regions where the approximation is still inaccurate.

The evaluation of the residual drives `bvp4c`'s adaptive mesh refinement procedure: after each Newton iteration, the solver estimates the error by studying the size and local variation of $r(x)$. Where the residual is large, the mesh is refined by inserting new points; where it is small, the mesh remains sparse. This mechanism enables `bvp4c` to converge reliably even when the initial mesh or initial guess is not accurate, and helps explain the solver's ability to resolve sharp gradients or rapidly varying solutions without excessive computational cost.

From the user's perspective, solving a BVP with `bvp4c` requires only three elements: a function defining the system of ODEs, a function describing the boundary conditions and an initial guess usually provided with `bvpinit`. The solver is then called via

```
sol = bvp4c(odefun,bcfun,solinit)
```

and returns a solution structure containing the mesh, the collocation polynomial $S(x)$ and its derivative. These can be evaluated at arbitrary points using `deval()`, allowing the user to visualize the solution and inspect its accuracy.

Thanks to the combination of fourth-order accuracy, residual-based adaptivity and robust Newton iterations, `bvp4c` is particularly effective for problems where the initial guess is poor and where the solution involves moderate nonlinearities or localized variations. Its reliability in these settings explains its widespread adoption in applied mathematics and engineering.

3.2 The bvp5c solver

The solver `bvp5c` is an higher-order extension of `bvp4c`, based on a collocation procedure employing the four-stage Lobatto IIIa formula. This method produces a piecewise polynomial approximation of degree four on each subinterval and achieves a global accuracy of order five. The increased order comes from the use of an additional interior collocation node compared to `bvp4c`, allowing the solver to extract more information from each subinterval of the mesh and thus reduce the overall discretization error.

As in `bvp4c`, the differential problem must first be reformulated as a system of first-order ODEs

$$y' = f(x,y,p), \quad a \le x \le b,$$

supplemented with boundary conditions

$$g(y(a),y(b),p) = 0.$$

Once the problem is specified, `bvp5c` constructs an approximation of the solution in the form of a continuously differentiable, piecewise polynomial function $S(x)$. In each subinterval, this polynomial is determined by requiring the differential equation to be satisfied at the Lobatto IIIa nodes of the four-stage formula, namely the two endpoints and two interior points. The resulting system is nonlinear and it is solved through Newton's method, exactly as in `bvp4c`.

The approximation produced by `bvp5c` is globally C^1 and the availability of a smooth derivative allows the solver to evaluate both the residual of the differential equation and more accurate estimates of the local error. A distinctive aspect of `bvp5c` is that its adaptive strategy is based on an estimate of the true local discretization error rather than on the size of the residual alone. The TLDE is the measure of the actual error introduced by a numerical method in a single integration step, representing the difference between the exact solution and the numerical approximation over that step. This difference comes from the higher-order structure of the Lobatto IIIa scheme used by `bvp5c` and allows the solver to obtain highly accurate solutions. As a consequence, `bvp5c` typically requires fewer mesh points than `bvp4c` to reach comparable accuracy, particularly when the solution is very smooth or the problem does not exhibit severe nonlinearities.

From the practical viewpoint, the user interacts with `bvp5c` in exactly the same way as with `bvp4c`. The ODE function, the boundary condition function, and an initial guess constructed via `bvpinit` are passed to the solver through

```
sol = bvp5c(odefun,bcfun,solinit).
```

The output is a structure containing the mesh, the collocation polynomial $S(x)$ and its derivative, all of which can be evaluated at arbitrary points using the `deval()` function. This uniform interface between the two solvers allows for direct comparison of their performance on the same set of problems.

The higher order of accuracy makes `bvp5c` particularly attractive for problems in which smoothness is expected and precision is important. However, its improved accuracy comes with a higher computational cost per iteration, due to the increased number of collocation conditions and the larger nonlinear systems that must be solved. For this reason, `bvp5c` is most effective when the mesh can remain relatively sparse. On problems that are stiff, highly nonlinear, or sensitive to the initial guess, `bvp4c` may still show superior robustness. Therefore, the two solvers complement one another: `bvp4c` excels in stability and tolerance to poor initial guesses, while `bvp5c` achieves high accuracy with fewer mesh points when the problem permits it.

3.3 Key differences

Although `bvp4c` and `bvp5c` share the same numerical philosophy - collocation based on Lobatto IIIa formulas, Newton's method for solving the resulting algebraic system and adaptive mesh refinement - their behavior in practice can differ substantially. The most immediate distinction concerns the order of accuracy: `bvp4c` achieves fourth-order convergence, while `bvp5c` reaches fifth-order thanks to the additional collocation stage. The higher order often allows `bvp5c` to obtain solutions of comparable accuracy with fewer mesh points, particularly when the exact solution is smooth and the problem does not exhibit strong nonlinearities or stiff components.

The two solvers also differ in how they assess the quality of the numerical approximation. In `bvp4c`, the adaptive refinement is guided by the residual of the collocation polynomial, whose smoothness makes it a reliable indicator even when the initial mesh is far from optimal. In contrast, `bvp5c` relies on a more direct estimate of the local discretization error. This distinction can influence their robustness: residual-based control tends to be more forgiving when the initial guess is poor or when the solution varies sharply, whereas direct error estimation is more efficient on well-behaved problems but may become sensitive in challenging regimes.

These differences reflect a fundamental trade-off. The higher order of `bvp5c` generally improves accuracy per mesh point but also increases the computational cost required at each iteration, due to the larger nonlinear systems associated with its four-stage collocation formula. As a result, `bvp4c` often remains the faster and more robust option for strongly nonlinear problems or for situations in which the initial guess is only approximate. Conversely, when the solution is sufficiently smooth and precision is essential, `bvp5c` may achieve the same accuracy with a significantly smaller discretization.

In summary, the two solvers should be regarded as complementary rather than interchangeable. Their contrasting strengths offer the user flexibility in selecting the most suitable method for the problem at hand.

4 Implementation and Software

To complement the theoretical study and the analysis of the two MATLAB solvers, we developed a graphical application using MATLAB App Designer. The goal of this tool is to provide an accessible environment for testing and comparing `bvp4c` and `bvp5c` on a range of representative boundary value problems. The interface is designed to guide the user through the solution process in a clear and intuitive manner, while offering immediate visual and quantitative feedback on the numerical performance of each solver.

4.1 Graphical User Interface (GUI) Workflow

The application is organized for a clear workflow. The user begins by selecting one of the test problems available in the app. All these problems are taken from the paper by Shampine, Kierzenka, and Reichelt (2000) (2), which is widely used as a benchmark for evaluating collocation-based BVP solvers. Once a problem is selected, the interface displays its mathematical formulation, including the differential equations, the boundary conditions and any parameters that may need to be determined as part of the solution.

After choosing the problem, the user selects which solver to employ. The interface offers both `bvp4c` and `bvp5c`, allowing a direct comparison of their behavior under the same conditions. The computation is initiated through a dedicated "Solve" button, after which the app calls the solver with the corresponding ODE function, boundary condition function and initial guess. When the computation is complete, the numerical solution is plotted on the main display panel and a summary of relevant performance indicators - such as the number of mesh points, the estimated error or residual, and the elapsed computation time - is shown in a table below the plot.

This dynamic workflow provides a clear and immediate understanding of how the two solvers differ in accuracy, mesh refinement and computational efficiency.

4.2 Code Organization

The software is structured in a modular fashion to ensure clarity, maintainability and extensibility. Each test problem is implemented in a separate MATLAB file that defines the mathematical components required by the solvers. These files include the function describing the system of first-order differential equations, typically written as

```
dydx = ode(x,y,p)
```

and the function implementing the boundary conditions,

```
res = bc(ya,yb,p).
```

When present unknown parameters p are incorporated directly into both functions allowing the solver to treat them as additional components of the algebraic system.

In several problems, an initial guess function is also provided, either as a vector of constants or as a more elaborate function of the independent variable. This guess is passed to `bvpinit`, which constructs the initial mesh and the corresponding approximation. In the second example, where an exact analytical solution is available, a dedicated function computes this reference solution so that the user can directly compare the true and numerical results.

The main App Designer file loads the appropriate problem definition based on the user's selection and passes the corresponding function handles to the chosen solver. This modular design facilitates the addition of new test problems and ensures that the interface remains clean and easy to navigate.

5 Case Studies and Analysis

5.1 BVP with periodic solution

As mentioned in the introduction, one of the most important aspects of `bvp4c` and `bvp5c` is their capability of dealing with non-separated boundary conditions, as shown in the Example 4 of the paper (2). The problem is defined by a system of two first-order ODEs in the variables y_1 and y_2, which describes the propagation of nerve impulses:

$$\begin{cases} y_1' = 3\left(y_1 + y_2 - \frac{1}{3}y_1^3 - 1.3\right) \\ y_2' = -\frac{(y_1 - 0.7 + 0.8y_2)}{3} \end{cases} \tag{4}$$

subject to periodic boundary conditions:

$$y_1(0) = y_1(T), \qquad y_2(0) = y_2(T) \tag{5}$$

where T in the unknown period. This is an example of free boundary problem, because one of the two endpoints of the domain where we solve the problem is unknown, in this case the right endpoint. To proceed with the solution we must prepare the system before submitting it to the solvers. We proceed by a change of variable, substituting t with $\tau = t/T$; hence, the differential equations become

$$\begin{cases} \frac{dy_1}{d\tau} = 3T\left(y_1 + y_2 - \frac{1}{3}y_1^3 - 1.3\right) \\ \frac{dy_2}{d\tau} = -T\frac{(y_1 - 0.7 + 0.8y_2)}{3} \end{cases} \tag{6}$$

The problem now is defined on a fixed interval with both the endpoints known, $[0,\ 1]$, and the non-separated boundary conditions are

$$y_1(0) = y_1(1), \qquad y_2(0) = y_2(1). \tag{7}$$

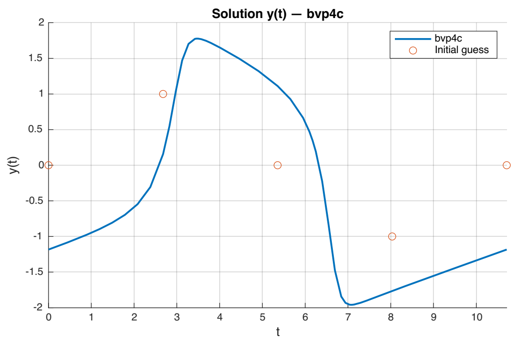
We impose another boundary condition to determine the unknown period T: this condition is chosen to eliminate all the degenerate solutions (with a constantly zero first derivative) and solutions with $T = 0$. This requirement, called *phase condition*, is

$$\frac{dy_2}{d\tau}(0) = 1, \qquad \text{i. e.} \quad -T\frac{((y_1(0) - 0.7 + 0.8y_2(0))}{3} = 1$$

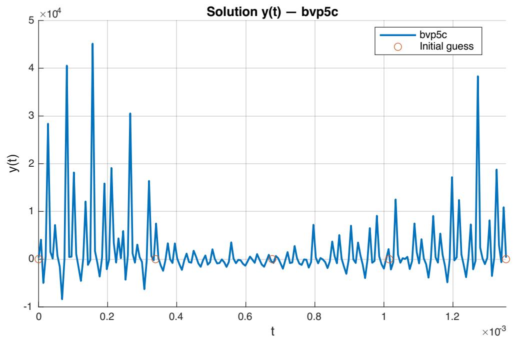
Since the desired solution is periodic, periodic functions are taken as initial guess

$$\tilde{y_1}(x) = sin(2\pi x), \qquad \tilde{y_2}(x) = cos(2\pi x)$$

while the initial guess for the period is chosen equal to 2π. These are the plots (rescaled to their original value $t = T\tau$) obtained by the two solvers:



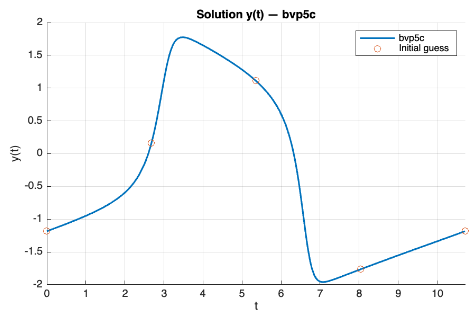
(a) Solution with `bvp4c`



(b) Solution with `bvp5c`

Figure 1: Approximated solutions

As we can see from the plots, the solution obtained with `bvp4c` is a good solution, despite the poor initial guess; while the one with `bvp5c` swings a lot, showing its instability, probably due to the poor initial guess. Indeed, to solve this problem and construct a good approximation, also with the 5th-order solver, we have constructed a better initial guess: we first used `bvp4c` to obtain a good approximation of the desired solution and then we gave this ”good” initial guess to `bvp5c` obtaining a much better result:



Hence, from these outcomes, we can observe that, despite `bvp5c`'s higher order, `bvp4c` is much more robust being able, even with a poor initial guess, to construct a good approximation of the solution. This robustness of `bvp4c` is probably due to its reliance on residual control, a strategy which provides a better error estimate than traditional discretization error control methods, especially when dealing with coarse initial meshes or poor solution approximations. The quantitative results further confirm this trade-off: `bvp5c` requires 973 mesh points to, unsuccessfully, process the poor initial guess. In contrast, `bvp4c` is much more robust, solving the problem successfully with 269 mesh points and approximately 0.04 seconds. However, once `bvp5c` is provided with a good initial guess, its superior efficiency becomes clear: it converges successfully using significantly fewer nodes — 107 mesh points — in approximately 0.08 seconds. Moreover, the final maximum residual confirms the control strategy of the two solvers: `bvp4c` achieves an ultimate residual of 8.8818×10^{-16}, a smaller value than the 3.5527×10^{-15} residual obtained by `bvp5c`, with the improved initial guess. In summary, `bvp4c` shows greater tolerance and stability for poor initial guesses, while `bvp5c` demonstrates significantly better mesh efficiency, requiring fewer points to achieve the solution, but only when given a sufficiently accurate starting point.

5.2 BVP with an infinite interval

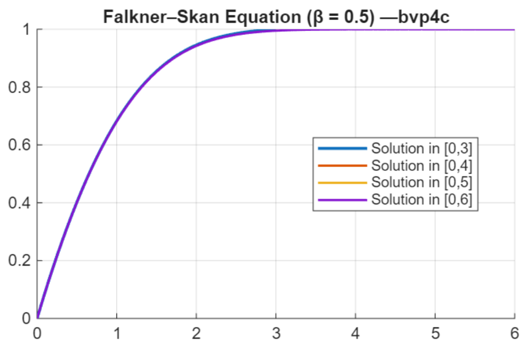
The Falkner-Skan problem is a boundary value problem (BVP) defined on an infinite interval, which originates from a similarity solution describing the viscous, incompressible, laminar flow over a flat plate. The governing differential equation is a third-order nonlinear ODE:

$$f''' + ff'' + \beta(1-(f')^2) = 0. \tag{8}$$

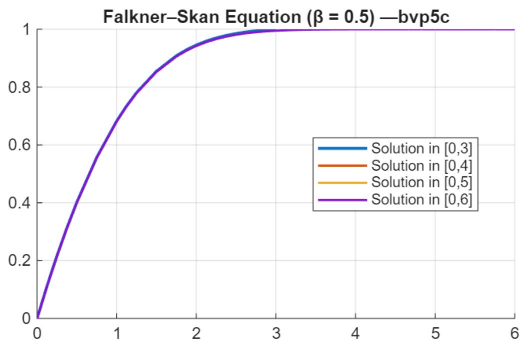
This problem is subject to boundary conditions $f(0) = 0$, $f'(0) = 0$ and $f'(\eta) \to 1$ as $\eta \to \infty$. The numerical solution of this BVP needs the application of several essential techniques. The primary challenge is the infinite domain, which is typically handled by replacing the boundary condition at infinity with one at a finite point. For instance, using $f'(6) = 1$ for the specific case where $\beta = 0.5$ is a good choice.

The second required step is the reformulation of the third-order ODE into a system of three first-order equations, a standard practice for numerical solvers like `bvp4c` or `bvp5c`, by introducing the auxiliary variables $y_2 = f'$ and $y_3 = f''$. The nature of the flow depends on the physical parameter β: for accelerating flows, i. e. $\beta > 0$, physically relevant solutions exist only for $-0.19884 \leq \beta \leq 2$, but we solved the equation for $\beta = 0.5$, which it's a relatively difficult case. We solved the problem on progressively larger intervals and the figures below show that it approaches 1 very rapidly as $\eta \to 6$, so taking 6 as "infinity" appears to be reasonable for this problem.

Finally, due to the problem's sensitivity and the difficulty in finding a good initial solution guess, the continuation method is employed. This technique consists in solving the problem at every iteration in the interval $[0,b]$, which is progressively extended, and the solution found in this interval it is used as initial guess for the following bigger intervals.



(a) Solution with `bvp4c`



(b) Solution with `bvp5c`

Figure 2: Approximated solutions

Now we can compare the two methods. As shown in the figures, the solutions produced by the two solvers are extremely close, essentially indistinguishable. Using `bvp4c`, the number of mesh points required for the successive solutions over the intervals $[0,3], [0,4], [0,5]$, and $[0,6]$ is $55, 91, 110$, and 117, respectively. In contrast, `bvp5c` achieves the same level of accuracy with only $21, 27, 33$, and 37 mesh points. For both solvers, the number of mesh points progressively increases as we consider increasingly large intervals. However, `bvp4c`, having a lower order of accuracy, requires a larger number of mesh points to keep the error within the prescribed tolerance. `bvp5c`, on the other hand, maintains the same accuracy with significantly fewer points, confirming its higher efficiency for problems where a higher-order method provides a clear advantage. Regarding the residual, when using `bvp4c` the solution over the first interval $[0,3]$ exhibits an extremely small residual of approximately 6.9389×10^{-18}. For the larger intervals, the residual stabilizes around 3.3307×10^{-16}. When using `bvp5c`, all solutions yield a residual of 4.4409×10^{-16}. The smaller residual obtained by `bvp4c` on the first interval can be explained by the fact that the Falkner–Skan solution behaves more regularly near 0, allowing the solver to converge very quickly. Over larger intervals, the problem becomes more challenging because the solver must handle a greater number of mesh points and the residual increases accordingly. In these intervals, the residual stabilizes around 3×10^{-16}, which reflects the convergence threshold. The residual produced by `bvp5c` is very close to the stabilized value of `bvp4c` and it remains essentially constant even on the smallest interval. Overall, the qualitative results indicate good convergence for both solvers, the small variations observed are due to numerical behavior and the limits of floating-point precision. The two solvers, `bvp4c` and `bvp5c`, achieved similarly high levels of accuracy in solving the Falkner-Skan boundary value problem, with both converging to a maximum residual error close to the hardware precision, approximately 1×10^{-16}. However, a comparison of their computational efficiency on the final interval $[0,6]$ reveals a notable difference in speed. The average execution time for `bvp4c` was 0.06874 seconds (with a standard deviation of 0.01162 s), while `bvp5c` completed the calculation in an average of only 0.03478 seconds (with a standard deviation of 0.01018 s). This makes the higher-order solver, `bvp5c`, approximately twice as fast as `bvp4c` for this problem. This increased speed is a direct consequence of the higher-order accuracy of `bvp5c` (5th order compared to 4th order). Although `bvp5c` has a higher computational cost per iteration due to using four collocation nodes, its ability to achieve the required accuracy with significantly fewer mesh points (37 points versus 117 points for `bvp4c`) drastically reduces the total number of steps and iterations needed, resulting in a superior overall performance time on smooth problems like this one.

5.3 BVP with a singular behavior at the origin

This 7th example brings our attention to another important property of the two solvers: their ability to solve BVPs whose solutions have a singularity. In this example, indeed, the boundary value problem is defined as

$$y'' = \frac{y^3 - y'}{2x}$$

with boundary conditions $y(0) = 0.1$ and $y(16) = 1/6$. This problem cannot be solved directly by numerical methods since the division by zero at $x = 0$ would generate an error. Therefore, to handle this singularity, the idea is to study the solution in a neighborhood of the origin using a series expansion. Hence, considering a "small" distance $d > 0$ from the origin, we develop the solution and its derivative as:

$$\begin{aligned} y(d) &= 0.1 + y'(0)\frac{\sqrt{d}}{10} + \frac{d}{100} + \dots \\ y'(d) &= \frac{y'(0)}{20\sqrt{d}} + \frac{1}{100} + \dots \end{aligned}$$

The unknown value $y'(0)$ is treated as an unknown parameter p. Thus, the problem is now solved on $[d,\ 16]$ and the two new boundary conditions are computed using the expansion of the solution and of its derivative evaluated at $x = d$. The remaining boundary condition at $x = 16$ is kept the same. The value d must be chosen properly: it needs to be as small as possible to provide an accurate approximation of the solution with just few terms of the series and, at the same time, it must be large enough to avoid numerical difficulties with the singularity. For solving the example with both solvers, we have used $d = 0.1$, while, as initial guess,

$p = 0.2$ for $y'(0)$ and the constant vector $[1\ 1]$ for the solution. To get the final plot across the entire original interval $[0,16]$, we have extended the solution back to the origin adding the initial value $y(0) = 0.1$ and the computed parameter $p = y'(0)$, obtaining these results

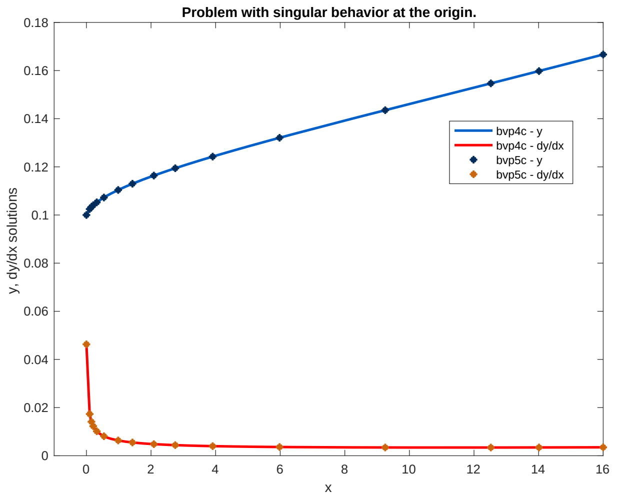


Figure 3: Solutions with `bvp4c` and `bvp5c`

From these plots, we can observe that both constructed solutions are visually almost identical, indicating a good reconstruction of the true solution by both solvers. However, the solution tables in the MATLAB App reveal one of the most significant differences between the two solvers: efficiency derived from the accuracy. Since `bvp5c` has an higher order than `bvp4c`, it allows us to obtain a good approximated solution, comparable to that of the lower-order solver, but using many fewer mesh points. Indeed, while `bvp4c` required 117 mesh points to construct the solution, `bvp5c` needed less than half that number, using only 44 points. However, as we expected, `bvp5c` is worse in computational time and residual, leading to slightly higher values in both metrics.

Therefore, these results highlight that `bvp5c` is the best choice for minimizing the number of mesh points while `bvp4c` remains the faster and more robust solver due to its lower computational cost per iteration.

In conclusion, this section illustrates a possible strategy for handling boundary value problems with singularities, whose implementation depends critically on the solution's local behavior. In *Porous Spherical Catalyst* example of the paper (2), the singularity was simple and permitted a direct approach by imposing a smooth condition at the origin. Conversely, this example was more challenging because the solution was not smooth near the singular point. This necessitated a more rigorous methodology: we used analytical series expansions to define the local behavior and subsequently re-mapped the solution to cover the entire integration domain.

5.4 Three-Point BVP

This example concerns a boundary value problem defined on an interval $[0,\lambda]$ with an emerging interior boundary condition at $x = 1$. Problems of this type cannot be solved directly with standard two-point solvers such as `bvp4c` and `bvp5c`, since these solvers require all boundary conditions to be imposed at the two endpoints of the computational domain. A reformulation is therefore necessary. The example considered here follows the classical strategy for handling multi-point conditions: the original problem is decomposed into two subproblems defined on adjacent intervals, which are then coupled through continuity conditions and mapped to a single domain. The original model consists of the coupled ODEs

$$v'(x) = \frac{C(x)-1}{n} \tag{9}$$

$$C'(x) = \frac{v(x)C(x) - \min(x,1)}{\eta} \tag{10}$$

with boundary conditions $v(0) = 0, C(\lambda) = 1$.
The term $min(x,1)$ in the equation for $C'(x)$ is not smooth at $x = 1$, so, following the standard procedure for multi-point BVPs, the interval is therefore partitioned into two regions: the first on $[0,1]$ and the second on $[1,\lambda]$, connected by the requirements that the functions $v(x)$ and $C(x)$ are continuous at $x = 1$ and here is where the BC at the interior point appears. To express both parts on a single computational interval $[0,1]$, the second region is mapped via the linear transformation

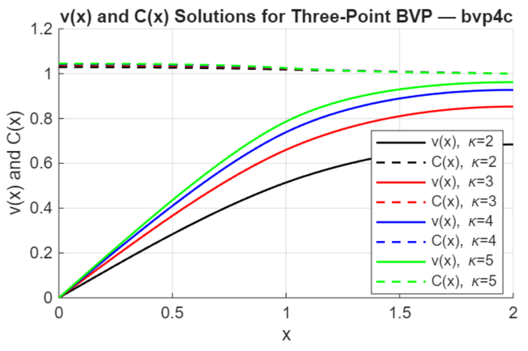
$$t = \frac{x-1}{\lambda-1}, \quad x = 1 + (\lambda-1)t.$$

The derivatives in the second region are accordingly rescaled by the factor $\lambda - 1$, giving rise to a system of four equations on $[0,1]$. If we denote by y_1 and y_2 the functions C and v on the first interval, and by y_3 and y_4 their counterparts on the second interval, the transformed system reads:

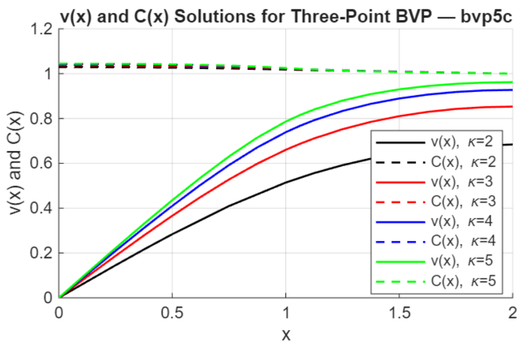
$$\begin{aligned} y_1' &= \frac{y_2-1}{\eta} \\ y_2' &= \frac{y_1 y_2 - x}{\eta} \\ y_3' &= \frac{(\lambda-1)(y_4-1)}{\eta} \\ y_4' &= \frac{(\lambda-1)(y_3 y_4-1)}{\eta}. \end{aligned}$$

The resulting conditions, $y_1(0) = 0$, $y_4(1) = 1$, $y_1(1) = y_3(0)$ and $y_2(1) = y_4(0)$, are non-separated boundary conditions, meaning they couple the solution components at both ends ($t = 0$ and $t = 1$). This final 4-component system on $t \in [0,1]$ is a standard two-point BVP perfectly suited for MATLAB's `bvp4c` and `bvp5c` solvers.

Because the solution depends on η, we consider the parameter κ such that $\eta = \lambda/(n\kappa^2)$ and the problem becomes increasingly challenging as κ varies. To guarantee convergence for all values in the prescribed range, the method of continuation is employed: the solution computed for a given value of κ is used as the initial guess for the next one. This strategy is particularly effective in multi-point problems, where a good initial guess strongly improves the robustness of the Newton iterations used internally by both `bvp4c` and `bvp5c`. The numerical results highlight the qualitative behavior of the system. The concentration $C(x)$ remains close to its final boundary value throughout the domain, reflecting a slow spatial variation, while the velocity $v(x)$, which starts from zero, increases monotonically with a slope that becomes steeper as κ grows.



(a) Solution with `bvp4c`



(b) Solution with `bvp5c`

Figure 4: Approximated solutions

Both solvers reproduce the qualitative behavior of the solution accurately, but their quantitative performance differs in a significant way. When continuation is applied across the full range of κ values, `bvp5c` achieves the prescribed accuracy using far fewer mesh points than `bvp4c`. In the continuation run reported in the Figure 4, `bvp4c` required 157 mesh points, whereas `bvp5c` reached the same level of accuracy with only 51 points, roughly one-third of the discretization needed by the lower-order solver. This reduction reflects the superior fifth-order accuracy of `bvp5c` and its ability to exploit the smoothness of the solution more efficiently. Despite requiring a denser mesh, `bvp4c` remained the faster solver overall. Thus, although `bvp5c` is markedly more efficient in terms of mesh refinement, `bvp4c` benefits from a lower computational overhead and delivers faster solutions in this specific problem. The residuals at the computed solutions are extremely small, with `bvp4c` achieving 1.1102×10^{-16} and `bvp5c` 4.4409×10^{-16}, confirming that both solvers reach a highly accurate approximation of the true solution.

These results emphasize how the structural differences between the solvers translate into practical performance. The higher order of `bvp5c` yields substantial savings in mesh density, while the residual-driven approach and lower algebraic complexity of `bvp4c` result in shorter execution times. For multi-point BVPs handled through domain transformations, the choice between the two solvers therefore depends on whether accuracy per mesh point or computational speed is the primary objective.

6 Conclusions

The project successfully analyzed the theoretical foundations and practical differences between the MATLAB boundary value problem solvers, `bvp4c` and `bvp5c`, using a custom Graphical User Interface (GUI) for experimentation.

The analysis conducted throughout the case studies highlights a clear trade-off between the two solvers. `bvp4c` proves to be more robust, particularly when the initial mesh or the initial guess is poor, or when the problem involves stiff or highly nonlinear dynamics. Its residual-based adaptivity ensures reliable convergence and an effective handling of local sharp variations. `bvp5c`, instead, thanks to its fifth-order accuracy, typically achieves comparable precision using significantly fewer mesh points. However, this improved efficiency comes with a higher computational cost per iteration and a greater sensitivity to initial guesses.

For this reason, `bvp4c` is generally preferable when computational reliability and robustness are the primary concerns, whereas `bvp5c` becomes advantageous when high accuracy is required and the solution is sufficiently smooth. The two solvers should be regarded as complementary tools, each suited to specific problem characteristics.

From a software development perspective, the design of the GUI provided a useful environment for exploring practically how solver performance depends on solution regularity and the quality of the initial approximation, offering immediate visual insight into the behavior of collocation-based methods.

Overall, the project underlines how theoretical understanding and computational experimentation can work together to guide the choice of the most effective numerical strategy for solving BVPs.

References

[1] The MathWorks Inc. *MATLAB: Boundary Value Problem Solvers (`bvp4c`, `bvp5c`)*. Natick, Massachusetts, 2024.

[2] Shampine, L.F., M.W. Reichelt, and J. Kierzenka. *Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with `bvp4c`*. MATLAB File Exchange, 2000.

[3] A. Spielauer and E.B. Weinmüller. *Numerical solution of boundary value problems in ordinary differential equations with time and space singularities*. Institute for Analysis and Scientific Computing, Vienna University of Technology, 2015.