

Towards a MATLAB Toolbox to compute backstepping kernels using the power series method

Xin Lin, Rafael Vazquez and Miroslav Krstic

Abstract—In this paper, we extend our previous work on the power series method for computing backstepping kernels. Our first contribution is the development of initial steps towards a MATLAB toolbox dedicated to backstepping kernel computation. This toolbox would exploit MATLAB’s linear algebra and sparse matrix manipulation features for enhanced efficiency; our initial findings show considerable improvements in computational speed with respect to the use of symbolical software without loss of precision at high orders. Additionally, we tackle limitations observed in our earlier work, such as slow convergence (due to oscillatory behaviors) and non-converging series (due to loss of analyticity at some singular points). To overcome these challenges, we introduce a technique that mitigates this behaviour by computing the expansion at different points, denoted as localized power series. This approach effectively navigates around singularities, and can also accelerate convergence by using more local approximations. Basic examples are provided to demonstrate these enhancements. Although this research is still ongoing, the significant potential and simplicity of the method already establish the power series approach as a viable and versatile solution for solving backstepping kernel equations, benefiting both novel and experienced practitioners in the field. We anticipate that these developments will be particularly beneficial in training the recently introduced neural operators that approximate backstepping kernels and gains.

I. INTRODUCTION

Backstepping has emerged as a powerful, versatile and broadly applicable technique for the control of systems governed by partial differential equations (PDEs). Initially introduced for the design of feedback laws and observers in one-dimensional reaction-diffusion PDEs [16], it has been extended to higher dimensions [26] and applied to a remarkably diverse array of applications, including flow control [23], [29], thermal convection loops [24], thermoacoustic instabilities [1], hyperbolic 1-D systems [7], [10], [15], multi-agent deployment [20], and wave equations [21]. Backstepping even allows for closed-form control laws in certain cases [25] and even the design of adaptive controllers [22].

The implementation of backstepping controllers and observers hinge on the solution of associated kernel PDEs. These are typically linear hyperbolic equations (first- or second-order) defined on a triangular (Goursat [11]) domain and subject to non-standard boundary conditions. The numerical solution of these equations presents a unique challenge,

especially for complex systems or those with discontinuous kernels, a common feature in coupled systems (both parabolic [27] and hyperbolic [12]); these discontinuities should be accurately captured with the numerical approximation if the implementation aims to be precise. While this topic is critical to backstepping applications, surprisingly, it has not received focused attention in the research literature.

Existing approaches for solving kernel equations include finite difference methods [2], [9], [14], [16], symbolic successive approximation series [24], or numerical solutions of integral forms of the equations [4], [13]. While some numerical methods exist for Goursat problems [8], they have not yet been explored in the context of backstepping kernels. These techniques can often lack flexibility, particularly when adapting to specific kernel equations or handling discontinuities.

The use of power series as an alternative method for solving backstepping kernel equations was introduced in several works. It was first proposed in [3], where convex optimization ideas were used to approximate kernels while maintaining stability, but without detailed analysis of convergence. Later, the method was used without proof in [6] for a problem involving coupled parabolic equations, where piecewise-analytic kernels required the use of several series to account for discontinuities. The first rigorous proof of uniqueness and convergence was given in [28], in the context of a multi-dimensional case with singularities at the origin, thus demonstrating both convergence and the kernel’s existence. Finally, in our recent work [30], we presented the power series-based method as a general approach, providing a solid theoretical foundation leveraging the classical successive approximation proof. We also introduced a framework for kernel computation based on Mathematica’s symbolic algebra capabilities. The advantages of the method (ease of implementation, accuracy, and the ability to produce parametric kernels) were highlighted, along with the primary limitation: the requirement of analyticity for system coefficients (satisfied by most backstepping applications). Slow convergence in some cases might necessitate high-order approximations, although modern computing power mitigates this issue at least for simple cases.

In this work, we significantly advance our previous power series method for backstepping kernel computation by developing a MATLAB-based implementation. Leveraging MATLAB’s sparse matrix manipulation and linear algebra features, we achieve substantial computational efficiency gains (of several orders of magnitude) while preserving the precision of our symbolic solver. This transition marks an initial step towards a user-friendly toolbox for the controls community, as reflected in the name given to this paper. The

Xin Lin is with the Department of Research Center of Satellite Technology, Harbin Institute of Technology, Harbin, 150080 China, e-mail: linxin29b@hit.edu.cn.

R. Vazquez is with the Department of Aerospace Engineering, Universidad de Sevilla, Camino de los Descubrimientos s.n., 41092 Sevilla, Spain, e-mail: rvazquez1@us.es.

M. Krstic is with the Department of Mechanical and Aerospace Engineering, University of California San Diego, La Jolla, CA 92093-0411, USA.

paper provides links to Live Scripts prepared by the authors, so that interested readers can already try and implement by themselves the examples provided in the paper, or even adapt the code to cases of interest to them.

Furthermore, we address limitations encountered in our earlier work, such as slow or non-convergent series due to singularities as shown in examples of [30]. We introduce a technique employing power series expansions calculated at strategically chosen points. This approach effectively helps in circumventing singularities and accelerates convergence through *localized* approximations, and this is directly shown by revisiting a problematic example of [30].

These advancements, combined with the inherent simplicity of the power series method, establish it as a powerful and versatile tool for solving backstepping kernel equations. The MATLAB implementation further enhances its accessibility and efficiency. This is particularly relevant within the emerging context of training neural operators for backstepping kernel approximation, where solution quality plays a paramount role. Current progress in this area demonstrate the enormous potential of neural operators to effectively learn backstepping gains and controllers while maintaining stability guarantees. This is reflected in the latest works, which include methods for bypassing traditional gain computations entirely [5], designing neural operator-based observers and controllers [17], extending the approach to more complex hyperbolic PDE systems [31], developing adaptive neural operator-based control designs [18], and even implementing gain scheduling techniques using neural operators [19]. The methodology presented in this paper can significantly accelerate the generation of large solution sets required for training the approximating DeepONets used in these works, thereby facilitating their adoption and further advancement in the field of neural operator-based control for backstepping systems.

The focus of this paper is twofold. Firstly, we present our current progress on migrating the computational framework from Mathematica to MATLAB. To this end, in Section II, the paper first focuses on a framework to pose the power series computation of an example problem as an (sparse) linear algebra problem easily solvable with MATLAB, and in Section III we review all the basic examples from our initial paper. Secondly, we analyze in Section IV an idea to overcome the main weakness of the method, namely slowly-convergent or even non-convergent series, by moving the point of expansion, getting farther away from singular points; an example shows how both ideas are blended together to provide a much improved solution both in terms of computational efficiency and by avoiding non-convergent behaviour. MATLAB code is provided separately by the authors in the form of Live Scripts. We finish in Section ?? with some concluding remarks and future lines of work.

II. TOWARDS A MATLAB TOOLBOX

Our previous paper [30] introduced the concept of a power series solution by an example. We use the same example to illustrate the basic framework developed for the MATLAB solver, even if at the cost of some redundancy, to make this paper self-contained.

A. A review of the basic results of [30]

Consider the reaction-diffusion equation,

$$u_t = \epsilon u_{xx} + \lambda(x)u, \quad (1)$$

for $t > 0$, with $\epsilon > 0$ and $\lambda(x)$ an analytic function in the domain $x \in [0, L]$, and with boundary conditions,

$$u(t, L) = U(t), \quad u(t, 0) = 0, \quad (2)$$

where U is the actuation variable. For sufficiently large $\lambda(x) > 0$, (1)–(2) becomes open-loop unstable.

Applying backstepping [16], one chooses a tuning parameter $c \geq 0$; then, the stabilizing control law is

$$U = \int_0^L K(L, \xi)u(\xi)d\xi, \quad (3)$$

where the kernel function $K(x, \xi)$ verifies the so-called *kernel equations*:

$$K_{xx}(x, \xi) - K_{\xi\xi}(x, \xi) = \frac{\lambda(\xi) + c}{\epsilon}K(x, \xi) \quad (4)$$

with the boundary conditions

$$K(x, x) = -\frac{1}{2\epsilon} \int_0^x (\lambda(\xi) + c) d\xi \quad (5)$$

$$K(x, 0) = 0 \quad (6)$$

in the *triangular* domain $\mathcal{T} = \{(x, \xi) : 0 \leq \xi \leq x \leq L\}$.

Denote \mathcal{D}_L as the complex-valued open disc centered at the origin and of radius L , i.e., $\mathcal{D}_L = \{x \in \mathbb{C} : |x| < L\}$. The following theorem holds [30]:

Theorem 1: If there exists $\delta > 0$ such that λ is analytic on $\mathcal{D}_{L+\delta}$, then the solution K of the kernel equations (4)–(6) can be extended to an analytic function in the polydisc $\mathcal{D}_{L+\delta/2} \times \mathcal{D}_{L+\delta/2}$ of \mathbb{C}^2 .

Note that such a $\delta > 0$ is required since it is essential to evaluate the kernel at $x = L$ for the control law (3). The only requirement of Theorem 1 in [30] is that there exists $\delta > 0$ such that λ is analytic on $\mathcal{D}_{L+\delta}$.

Based on Theorem 1, it is known that there exists a kernel function $K(x, \xi)$, analytic in both variables, which can therefore be expressed by using a *double* power series,

$$K(x, \xi) = \sum_{i=0}^{\infty} \sum_{j=0}^i K_{ij} x^{i-j} \xi^j \quad (7)$$

where K_{ij} denotes the series coefficients. In practice, an N -order truncation operator $T^N(\cdot)$ is defined for the power series to retain the terms of $x^i \xi^j$ with $i + j \leq N$ and discard the other terms, where N indicates the series truncation order. Applying the N -order truncation operator to (7) gives an N -order truncated approximation of the kernel function,

$$K^N(x, \xi) := T^N(K(x, \xi)) = \sum_{i=0}^N \sum_{j=0}^i K_{ij} x^{i-j} \xi^j \quad (8)$$

To compute (8), one starts by expressing the analytic function $[\lambda(\xi) + c]/\epsilon$ as a power series and applying the N -order truncation operator,

$$\frac{\lambda^N(\xi) + c}{\epsilon} := T^N \left(\frac{\lambda(\xi) + c}{\epsilon} \right) = \sum_{i=0}^N \lambda_i \xi^i \quad (9)$$

Substituting (8) and (9) into (4)–(6), and equating all terms having the same powers of x and ξ , one can obtain equations with respect to the coefficient K_{ij} (see (12)–(16) in [30]).

B. A MATLAB double series vector-matrix framework: definitions and operators

Up to here, the development mimics the one given in the previous paper [30], where the task of finding the coefficients is accomplished by using the symbolic capabilities of Mathematica. The symbolic procedure is fast enough for small orders, but the computation slows down when the size of equations becomes relatively high (which happens when the series truncation order N is large). To overcome this difficulty, a *vector-matrix formulation* of the problem is proposed to exploit MATLAB's linear algebra and sparse matrix manipulation features for much enhanced efficiency. The basic framework is explained below.

The N -order approximated polynomial $K^N(x, \xi)$ of the kernel function given in (8) is written in a vector form,

$$K^N(x, \xi) = \sum_{i=0}^N \sum_{j=0}^i K_{ij} x^{i-j} \xi^j := \boldsymbol{\kappa}^N \mathbf{z}^N \quad (10)$$

where $\boldsymbol{\kappa}^N = [K_{00}, K_{10}, K_{11}, \dots, K_{N0}, \dots, K_{NN}]$ and $\mathbf{z}^N = [x^0 \xi^0, x^1 \xi^0, x^0 \xi^1, \dots, x^N \xi^0, \dots, x^0 \xi^N]^T$. It can be seen that K_{ij} and $x^{i-j} \xi^j$ are respectively the $m(i, j)$ -th term of $\boldsymbol{\kappa}^N$ and \mathbf{z}^N , i.e., $\boldsymbol{\kappa}^N(m(i, j)) = K_{ij}$ and $\mathbf{z}^N(m(i, j)) = x^{i-j} \xi^j$, where $m(i, j) := l(i-1) + j + 1$, and where $l(i) := (i+1)(i+2)/2$. Thus, these functions m and l translate indexes of the double series to indexes of the coefficient vector.

The next step is to appropriately define transformation matrices so that the different operators that one needs to apply to $K^N(x, \xi)$ (truncation, differentiation, trace operators at the boundaries, multiplication by $\lambda(\xi)$) are transformed into a matrix operation. The general mathematical formulation of such operators is

$$f(K^N(x, \xi)) = \boldsymbol{\kappa}^N \mathbf{R}^N \mathbf{z}^N \quad (11)$$

where $f(\cdot)$ denotes any operator applied to $K^N(x, \xi)$, $\mathbf{R}^N \in \mathbb{R}^{l(N) \times l(N)}$ denotes the corresponding transformation matrix.

Based on (10), the operation of taking a first-order partial derivative of $K^N(x, \xi)$ with respect to the first variable x is expressed as a truncated power series in the following way:

$$\begin{aligned} K_x^N(x, \xi) &= \sum_{i=0}^{N-1} \sum_{j=0}^i (i-j+1) K_{(i+1)j} x^{i-j} \xi^j \\ &:= \boldsymbol{\kappa}^N \mathbf{D}_x^N \mathbf{z}^N \end{aligned} \quad (12)$$

where $\mathbf{D}_x^N \in \mathbb{R}^{l(N) \times l(N)}$ is the transformation matrix for the operation of taking the first-order partial derivative with respect to x . Similarly, the transformation matrix \mathbf{D}_ξ^N for the first-order partial derivative $K_\xi^N(x, \xi)$ as well as the matrices \mathbf{D}_{xx}^{N-2} , $\mathbf{D}_{\xi\xi}^{N-2}$ for the second-order partial derivatives $K_{xx}^N(x, \xi)$, $K_{\xi\xi}^N(x, \xi)$ can be also obtained. Then, the $(N-2)$ -order truncation approximation of the left-side of the kernel equation (4) is given by

$$T^{N-2}(K_{xx} - K_{\xi\xi}) = \boldsymbol{\kappa}^N (\mathbf{D}_{xx}^{N-2} - \mathbf{D}_{\xi\xi}^{N-2}) \mathbf{z}^{N-2} \quad (13)$$

In addition, substituting (7) and (9) into the right-side of the kernel equation (4) and applying the $(N-2)$ -order truncation operator yields

$$\begin{aligned} &T^{N-2} \left(\frac{\lambda(\xi) + c}{\epsilon} K \right) \\ &= \sum_{i=0}^{N-2} \sum_{j=0}^i \left(\sum_{q=0}^j \lambda_q K_{(i-q)(j-q)} \right) x^{i-j} \xi^j \\ &:= \boldsymbol{\kappa}^N \mathbf{H}_{\xi, \lambda}^{N-2} \mathbf{z}^{N-2} \end{aligned} \quad (14)$$

where the transformation matrix $\mathbf{H}_{\xi, \lambda}^{N-2} \in \mathbb{R}^{l(N) \times l(N-2)}$.

Hence, for (4), one can obtain that

$$\boldsymbol{\kappa}^N \mathbf{M}_K \mathbf{z}^{N-2} = 0 \quad (15)$$

where $\mathbf{M}_K = \mathbf{D}_{xx}^{N-2} - \mathbf{D}_{\xi\xi}^{N-2} - \mathbf{H}_{\xi, \lambda}^{N-2}$. Note that $\boldsymbol{\kappa}^N \mathbf{M}_K$ are the coefficients of a power series which equals zero, and therefore must be zero themselves. Thus one can write

$$\boldsymbol{\kappa}^N \mathbf{M}_K = \mathbf{b}_K = \mathbf{0}_{1 \times l(N-2)} \quad (16)$$

For the boundary conditions (5) and (6), mimicking the above development process, one can get

$$\begin{cases} (\boldsymbol{\kappa}^N \mathbf{P}_1^N - \boldsymbol{\lambda}_{\text{int}}^N) \mathbf{x}^N = 0 \\ (\boldsymbol{\kappa}^N \mathbf{P}_0^N) \mathbf{x}^N = 0 \end{cases} \Rightarrow \begin{cases} \boldsymbol{\kappa}^N \mathbf{P}_1^N = \boldsymbol{\lambda}_{\text{int}}^N \\ \boldsymbol{\kappa}^N \mathbf{P}_0^N = \mathbf{0}_{1 \times (N+1)} \end{cases} \quad (17)$$

where $\mathbf{x}^N = [x^0, \dots, x^N]^T$, $\boldsymbol{\lambda}_{\text{int}} = [0, \frac{-\lambda_0}{2}, \dots, \frac{-\lambda_{N-1}}{2N}]$, $\mathbf{P}_\alpha^N \in \mathbb{R}^{l(N) \times (N+1)}$ denotes the transformation matrix of the operation of taking the second variable ξ in $K^N(x, \xi)$ as a function of the first variable x , i.e., $\xi = \alpha x$. In addition, note that the first equations of the two systems of equations in (17) are the same, both being $K_{00} = 0$, thereby one of them can be discarded. Therefore, (17) can be written as

$$\boldsymbol{\kappa}^N \mathbf{M}_{BC,1} = \mathbf{b}_{BC,1} = \boldsymbol{\lambda}_{\text{int}}^N \quad (18)$$

$$\boldsymbol{\kappa}^N \mathbf{M}_{BC,2} = \mathbf{b}_{BC,2} = \mathbf{0}_{1 \times N} \quad (19)$$

where $\mathbf{M}_{BC,1} = \mathbf{P}_1^N$ and $\mathbf{M}_{BC,2}$ includes the second to the last column of \mathbf{P}_0^N .

Finally, based on (16), (18), and (19), the equations with respect to the series coefficient vector $\boldsymbol{\kappa}^N$ are obtained as

$$\boldsymbol{\kappa}^N \bar{\mathbf{M}} = \bar{\mathbf{b}} \quad (20)$$

where

$$\bar{\mathbf{M}} = [\mathbf{M}_K, \mathbf{M}_{BC,1}, \mathbf{M}_{BC,2}], \bar{\mathbf{b}} = [\mathbf{b}_K, \mathbf{b}_{BC,1}, \mathbf{b}_{BC,2}] \quad (21)$$

(20) is a system of linear equations with respect to $\boldsymbol{\kappa}^N$, and it is easy to verify that the numbers of unknown variables and equations are the same, both being $l(N)$. The linear equations of $\boldsymbol{\kappa}^N$ can be efficiently solved with MATLAB.

In addition, when the series truncation order N becomes large, it can be seen that most elements of the transformation matrices \mathbf{D}_x^N , \mathbf{D}_ξ^N , and $\mathbf{H}_{\xi, \lambda}^{N-2}$ will be 0, which implies that the system matrix $\bar{\mathbf{M}}$ in (20) may be sparse. In this case, the sparse matrix manipulation features in MATLAB can be adopted to enhance the solving efficiency by reducing the memory occupied by these matrices.

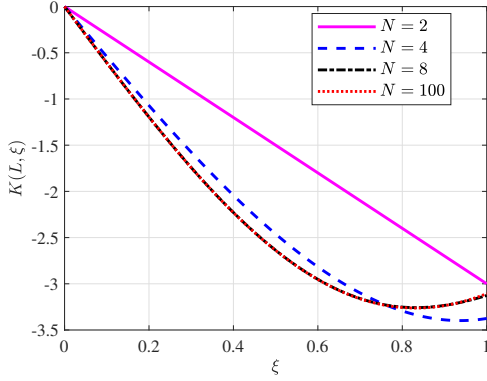


Fig. 1. Convergent case with $\lambda(x) = 3 + x^2 \sin(3x)$ for Example 1.

TABLE I

MATHEMATICA VS. MATLAB COMPUTATION TIMES (EXAMPLE 1).

N	Computation time (s)			Sparsity
	Mathematica	MATLAB		
		Non-sparse	Sparse	
2	2.03e-03	7.44e-05	7.82e-05	0.694
4	6.94e-03	8.73e-05	1.11e-04	0.836
6	1.66e-02	1.19e-04	1.23e-04	0.895
8	3.78e-02	1.33e-04	2.17e-04	0.926
25	2.43	4.23e-03	2.56e-03	0.982
50	176.49	8.07e-02	1.64e-02	0.992
100	—	1.83	1.14e-01	0.996

III. BASIC EXAMPLES REVISITED

Our previous paper [30] demonstrated the performance of the proposed power series method by some basic examples, which are solved by using the Mathematica solver [32]. In this section, we revisit these examples with the developed MATLAB solver, except for a symbolic example. To avoid redundancy, we only repeat the kernel equations but not the plant equations and plot just the kernel gains and use different values of the coefficients except for Example 1 which is used for computational speed comparison. The examples can be downloaded as Live Scripts¹, which are fully commented and can be reused by interested readers for their own kernel computations. Some examples in this section and the next provide computation times for comparison; all the tests are performed on an Intel[®] Core[™] i5-6300HQ 2.30GHz with Windows 10 and run on MATLAB 2018b.

A. Example 1: Parabolic equation with space-varying reaction

Consider $\lambda(x) = 3 + x^2 \sin(3x)$, $L = \epsilon = 1$, $c = 3$. This verifies Theorem 1 in [30] and Fig. 1 shows the resulting kernel gain for several orders of approximation by using the proposed MATLAB solver. It can be seen from Fig. 1 that the obtained results by the proposed MATLAB solver are the same as those by the Mathematica solver in [30]. The computation times of the Mathematica solver and the

MATLAB solver (with and without using sparse matrix definitions) for several series approximation orders are listed in Table I, where the results of the Mathematica solver for the case with $N = 100$ is not provided since it takes too long. The results of Table I clearly show a very significant computational efficiency advantage of the MATLAB solver relative to the Mathematica solver, particularly for large orders. The Mathematica solver shows sufficient computational speed only when N is small (e.g., $N \leq 8$), whereas the developed MATLAB solver is still fast enough even with a large N (e.g., $N = 100$). In addition, the sparsity of the problem for different series approximation orders are also listed in Table I, where the sparsity is defined as the ratio of the number of zero coefficients of the system matrix \bar{M} in (20) to the number of all elements of \bar{M} . The results of Table I show that when the sparsity of the problem is high (e.g., the cases with $N = 50$ and $N = 100$), the use of the sparse matrix manipulation features in MATLAB can enhance the computational efficiency (as well as memory savings, not shown).

B. Example 2: Motion planning kernels for $(0+2) \times (0+2)$ 1-D linear hyperbolic system with space-varying coupling

This was Example 5 in [30] and directly extracted from [12], having even a explicit solution for constant coefficients, which is the case under consideration. In this case one needs to find five gain kernels from L_{11} , L_{12} , L_{21} and L_{22} , that verify

$$\mu_1 \partial_x L_{11}(x, \xi) + \mu_1 \partial_\xi L_{11}(x, \xi) = \sigma_{21}(\xi) L_{12}(x, \xi), \quad (22)$$

$$\mu_1 \partial_x L_{12}(x, \xi) + \mu_2 \partial_\xi L_{12}(x, \xi) = \sigma_{12}(\xi) L_{11}(x, \xi), \quad (23)$$

$$\mu_2 \partial_x L_{21}(x, \xi) + \mu_1 \partial_\xi L_{21}(x, \xi) = \sigma_{21}(\xi) L_{22}(x, \xi), \quad (24)$$

$$\mu_2 \partial_x L_{22}(x, \xi) + \mu_2 \partial_\xi L_{22}(x, \xi) = \sigma_{12}(\xi) L_{21}(x, \xi), \quad (25)$$

with boundary conditions

$$L_{11}(x, 0) = L_{12}(x, 0) = L_{22}(x, 0) = 0, \quad (26)$$

$$L_{12}(x, x) = \frac{\sigma_{12}(x)}{\mu_2 - \mu_1}, L_{21}(x, x) = \frac{\sigma_{21}(x)}{\mu_1 - \mu_2}. \quad (27)$$

Here, $\mu_1 > \mu_2 > 0$ are the (constant) transport speeds. As explained in [12], L_{12} , differently from the other kernels, possesses two boundary conditions, namely $L_{12}(x, 0) = 0$ and $L_{12}(x, x) = \frac{\sigma_{12}(x)}{\mu_2 - \mu_1}$, which is solved by defining a

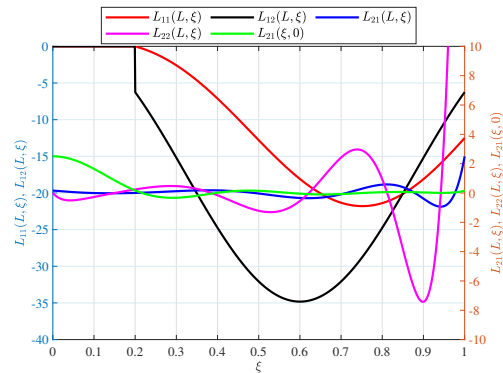


Fig. 2. Solutions of gain kernels for Example 2. Note the discontinuous kernel $L_{12}(L, \xi)$ and the piecewise differentiable kernel $L_{11}(L, \xi)$.

¹The MATLAB Live Scripts can be downloaded from the website <http://aero.us.es/rvazquez/PowerSeriesLiveScripts.zip>.

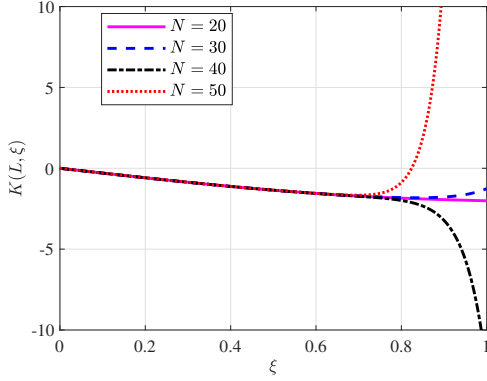


Fig. 3. Divergent case with $\lambda(x) = \sqrt{0.5 + x^2}$ (Example 1).

piecewise smooth kernels; because of the coupling with L_{11} this necessitates a similar piecewise definition for L_{11} .

Consider the particular case $\mu_1 = 1$, $\mu_2 = 0.2$, $\sigma_{12} = 5$, and $\sigma_{21} = 2$ considered in [12]. The example presents numerical difficulties due to oscillations for these chosen coefficients and the Mathematica symbolic solver of [30] takes too long to obtain an accurate solution. To mitigate the problem, note that the kernels (L_{11}, L_{12}) and (L_{21}, L_{22}) can be computed individually, thereby the series truncation orders can be selected individually, which are denoted as N_{L_1} for (L_{11}, L_{12}) and N_{L_2} for (L_{21}, L_{22}) . The resulting gain kernels are shown in Fig. 2 for $N_{L_1} = 8$ and $N_{L_1} = 40$, which shows that a small order is enough for (L_{11}, L_{12}) and a relatively large order is needed for (L_{21}, L_{22}) (which are more oscillatory). The obtained results are consistent with those by the explicit solutions given in [12] (see Fig. 1 in that paper), and the computation time is 1.93e-02 s.

IV. CHANGING THE POINT OF EXPANSION TO ACCELERATE CONVERGENCE AND AVOID SINGULARITIES

For simplicity, the arguments of this section refer to the basic example (4)–(6). According to Theorem 1 in [30], a $\delta > 0$ needs to exist such that $\lambda(x)$ is analytic on the disc $\mathcal{D}_{L+\delta}$ to ensure the convergence of the power series solution in (7). When the function is not analytic within the disc $\mathcal{D}_{L+\delta}$, the power series solution diverges. For example, if one has $\lambda(x) = \sqrt{0.5 + x^2}$ which is not analytic on the unit disc, the kernel gain does not converge as the order increases as it can clearly be seen in the Fig. 3 by inspecting the gain for different orders of the series. In this section, we will show that for some cases the divergent problem can be resolved by expanding (*localizing*) the series at a different point. We call this new power series, expanded at a strategically-chosen point, a “localized” power series.

Now, the kernel $K(x, \xi)$ is expressed by using a *double* power series around the point (x_0, ξ_0) rather than the origin,

$$K(x, \xi) = \sum_{i=0}^{\infty} \sum_{j=0}^i K_{ij} (x - x_0)^{i-j} (\xi - \xi_0)^j \quad (28)$$

Denote $\tilde{x} = x - x_0$ and $\tilde{\xi} = \xi - \xi_0$. Then, (28) becomes

$$\tilde{K}(\tilde{x}, \tilde{\xi}) = \sum_{i=0}^{\infty} \sum_{j=0}^i K_{ij} \tilde{x}^{i-j} \tilde{\xi}^j \quad (29)$$

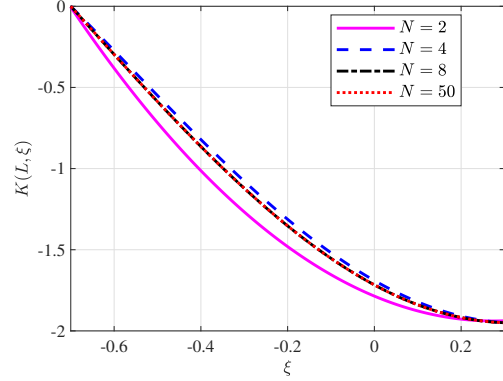


Fig. 4. Convergent case with $\lambda(x) = \sqrt{0.5 + x^2}$ by localized power series (Example 1).

Based on (29), the *kernel equations* in (4) can be transformed into those in terms of \tilde{x} and $\tilde{\xi}$,

$$\tilde{K}_{\tilde{x}\tilde{x}}(\tilde{x}, \tilde{\xi}) - \tilde{K}_{\tilde{\xi}\tilde{\xi}}(\tilde{x}, \tilde{\xi}) = \frac{\lambda(\tilde{\xi} + \xi_0) + c}{\epsilon} \tilde{K}(\tilde{x}, \tilde{\xi}) \quad (30)$$

For the left-sides of boundary conditions in (5) and (6), the values of the second variable ξ are respectively $\xi = x$ for the first boundary condition and $\xi = 0$ for the second. Thus, in terms of \tilde{x} and $\tilde{\xi}$, they result in $\tilde{\xi} = \tilde{x} + x_0 - \xi_0$ for the first boundary condition and $\tilde{\xi} = -\xi_0$ for the second. For the integral in the right-side of boundary condition in (5), it can be also expressed in terms of \tilde{x} and $\tilde{\xi}$,

$$\int_0^x (\lambda(\xi) + c) d\xi = \int_{-\xi_0}^{\tilde{x} + x_0 - \xi_0} (\lambda(\tilde{\xi} + \xi_0) + c) d\tilde{\xi} \quad (31)$$

Based on the change of variables and (31), the boundary conditions (5) and (6) can be expressed in terms of \tilde{x} and $\tilde{\xi}$,

$$\tilde{K}(\tilde{x}, \tilde{x}_1) = \frac{-1}{2\epsilon} \int_{-\xi_0}^{\tilde{x}_1} (\lambda(\tilde{\xi} + \xi_0) + c) d\tilde{\xi} \quad (32)$$

$$\tilde{K}(\tilde{x}, -\xi_0) = 0 \quad (33)$$

where $\tilde{x}_1 = \tilde{x} + x_0 - \xi_0$.

For the transformed kernel equations in (30) and the boundary conditions in (32) and (33), a Theorem similar to Theorem 1 in [30] can be stated for the given x_0 and ξ_0 :

Theorem 2 (Localized power series): If there exists $\delta > 0$ such that $\lambda(\xi)$ is analytic on the disc $\mathcal{D}_{L+\delta}(\xi_0)$ centered at ξ_0 , and such disc covers the interval $[0, 1]$, then there exists a x_0 and power series solution in the form of (28) which converges and defines an analytic function in the polydisc $\mathcal{D}_{L+\delta/2}(x_0) \times \mathcal{D}_{L+\delta/2}(\xi_0)$, that is the unique solution of the kernel equations (4)–(6).

The proof of the above Theorem is similar to Theorem 1 in [30], since only the point of expansion varies and in [30] it was shown that the kernel is analytic in \mathcal{T} ; by uniqueness of analytic functions any converging power series representing the kernel has to solve the kernel equations. In principle one can choose $x_0 = \xi_0$ but other values are admissible. This theorem can be easily adapted to other examples, which is not done here due to lack of space. Instead, some numerical evidence is provided next for the basic example.

TABLE II

ORIGIN-BASED (DIVERGENT) VS. LOCALIZED (CONVERGENT)

N	POWER SERIES			
	Computation time (s)		Sparsity	
	At origin	Localized	At origin	Localized
2	8.08e-05	1.08e-04	0.694	0.556
4	1.12e-04	1.68e-04	0.831	0.702
6	1.68e-04	2.79e-04	0.890	0.783
8	2.44e-04	2.15e-03	0.921	0.832
15	4.98e-04	1.27e-03	0.964	0.908
25	1.32e-03	3.32e-03	0.981	0.945
50	6.37e-03	2.49e-02	0.992	0.973

A. Example 3: Parabolic equation with space-varying reaction not verifying Theorem 1 in [30]

Choosing $x_0 = 0.5$ and $\xi_0 = 0.7$, it can be verified that the above Theorem is satisfied for $\lambda(x) = \sqrt{0.5 + x^2}$, and the resulting kernel gain for several orders of approximation by using the proposed MATLAB solver is plotted in Fig. 4, which clearly shows that the localized power series solution is convergent. In addition, the sparsities and computation times of the zero-centered solution (divergent solution) and the localized solution (convergent solution) are listed in Table. II, which shows that convergence is achieved at the cost of losing some sparsity and computational speed; the loss is in any case reasonable.

The idea of a localized power series gives rise to an additional concept, that of *patches* of power series, where instead of a single power series representation one employs several localized approximations. This would allow to better approximate oscillatory kernels while at the same time using smaller orders of approximation, for instance such an idea could reduce the order required to solve the Example 4 of Section III. This concept will be explored in future work.

ACKNOWLEDGMENTS

Rafael Vazquez acknowledges support by grant TED2021-132099B-C33 funded by MICIU/AEI/10.13039/501100011033 and by “European Union NextGenerationEU/PRTR.” The work is partially supported by the China Scholarship Council.

REFERENCES

- [1] G. Andrade, R. Vazquez, and D. Pagano, “Backstepping stabilization of a linearized ODE-PDE Rijke tube model,” *Automatica*, vol. 96, pp. 98–109, 2018.
- [2] H. Anfinsen and O. Aamo, *Adaptive control of hyperbolic PDEs*. New York: Springer, 2019.
- [3] P. Ascencio, A. Astolfi, and T. Parisini, “Backstepping PDE design: A convex optimization approach,” *IEEE Transactions on Automatic Control*, vol. 63, pp. 1943–1958, 2018.
- [4] J. Auriol and D. Bresch-Pietri, “Robust state-feedback stabilization of an underactuated network of interconnected $n + m$ hyperbolic PDE systems,” *Automatica*, vol. 136, p. 110040, 2022.
- [5] L. Bhan, Y. Shi, and M. Krstic, “Neural operators for bypassing gain and control computations in PDE backstepping,” *arXiv preprint arXiv:2302.14265*, 2023.
- [6] L. Camacho-Solorio, R. Vazquez, and M. Krstic, “Boundary observers for coupled diffusion-reaction systems with prescribed convergence rate,” *Systems and Control Letters*, vol. 134, p. 104586, 2020.

- [7] J.-M. Coron, R. Vazquez, M. Krstic, and G. Bastin, “Local exponential H^2 stabilization of a 2×2 quasilinear hyperbolic system using backstepping,” *SIAM J. Control Optim.*, vol. 51, pp. 2005–2035, 2013.
- [8] J. Day, “A Runge-Kutta method for the numerical solution of the Goursat problem in hyperbolic partial differential equations,” *The Computer Journal*, vol. 9(1), pp. 81–83, 1966.
- [9] J. Deutscher and J. Gabriel, “A backstepping approach to output regulation for coupled linear wave-ODE systems,” *Automatica*, vol. 123, p. 109338, 2021.
- [10] F. Di Meglio, R. Vazquez, and M. Krstic, “Stabilization of a system of $n+1$ coupled first-order hyperbolic linear PDEs with a single boundary input,” *IEEE Trans. Aut. Contr.*, vol. 58, pp. 3097–3111, 2013.
- [11] R. Holten, “Generalized Goursat problem,” *Pacific Journal of Mathematics*, vol. 12, pp. 207–224, 1962.
- [12] L. Hu, F. Di Meglio, R. Vazquez, and M. Krstic, “Control of homodirectional and general heterodirectional linear coupled hyperbolic PDEs,” *IEEE Transactions on Automatic Control*, vol. 61, no. 10, pp. 3301–3314, 2016.
- [13] L. Jadachowski, T. Meurer, and A. Kugi, “An efficient implementation of backstepping observers for time-varying parabolic PDEs,” *IFAC Proceedings Volumes*, vol. 45, pp. 798–803, 2012.
- [14] S. Kerschbaum and J. Deutscher, “Backstepping control of coupled linear parabolic PDEs with space and time dependent coefficients,” *IEEE Trans. Aut. Contr.*, vol. 65(7), pp. 3060–3067, 2019.
- [15] M. Krstic and A. Smyshlyayev, “Backstepping boundary control for first order hyperbolic PDEs and application to systems with actuator and sensor delays,” *Syst. Contr. Lett.*, vol. 57, pp. 750–758, 2008.
- [16] —, *Boundary Control of PDEs*. SIAM, 2008.
- [17] M. Krstic, L. Bhan, and Y. Shi, “Neural operators of backstepping controller and observer gain functions for reaction-diffusion PDEs,” *arXiv preprint arXiv:2303.10506*, 2023.
- [18] M. Lamarque, L. Bhan, Y. Shi, and M. Krstic, “Adaptive neural-operator backstepping control of a class of nonlinear time-varying PDEs,” *arXiv preprint arXiv:2401.07521*, 2024.
- [19] M. Lamarque and M. Krstic, “Gain scheduling with a neural operator for a class of nonlinear time-varying PDEs,” *arXiv preprint arXiv:2401.02511*, 2024.
- [20] J. Qi, R. Vazquez, and M. Krstic, “Multi-agent deployment in 3-D via PDE control,” *IEEE Transactions on Automatic Control*, vol. 60, pp. 891–906, 2015.
- [21] A. Smyshlyayev, E. Cerpa, and M. Krstic, “Boundary stabilization of a 1-D wave equation with in-domain antidamping,” *SIAM J. Control Optim.*, vol. 48, pp. 4014–4031, 2010.
- [22] A. Smyshlyayev and M. Krstic, *Adaptive Control of Parabolic PDEs*. Princeton University Press, 2010.
- [23] R. Vazquez and M. Krstic, *Control of Turbulent and Magnetohydrodynamic Channel Flow*. Birkhauser, 2008.
- [24] —, “Boundary observer for output-feedback stabilization of thermal convection loop,” *IEEE Trans. Control Syst. Technol.*, vol. 18, pp. 789–797, 2010.
- [25] —, “Marcum Q-functions and explicit kernels for stabilization of linear hyperbolic systems with constant coefficients,” *Systems & Control Letters*, vol. 68, pp. 33–42, 2014.
- [26] —, “Boundary control of reaction-diffusion PDEs on balls in spaces of arbitrary dimensions,” *ESAIM:Control Optim. Calc. Var.*, vol. 22, no. 4, pp. 1078–1096, 2016.
- [27] —, “Boundary control of coupled reaction-advection-diffusion systems with spatially-varying coefficients,” *IEEE Transactions on Automatic Control*, vol. 62, pp. 2026–2033, 2017.
- [28] R. Vazquez, M. Krstic, J. Zhang, and J. Qi, “Kernel well-posedness and computation by power series in backstepping output feedback for radially-dependent reaction-diffusion PDEs on multidimensional balls,” *Systems & Control Letters*, vol. 177, p. 105538, 2023.
- [29] R. Vazquez, E. Trelat, and J.-M. Coron, “Control for fast and stable laminar-to-high-Reynolds-numbers transfer in a 2d Navier-Stokes channel flow,” *Disc. Cont. Dyn. Syst. Ser. B*, vol. 10, pp. 925–956, 2008.
- [30] R. Vazquez, G. Chen, J. Qiao, and M. Krstic, “The power series method to compute backstepping kernel gains: Theory and practice,” in *2023 62nd IEEE Conference on Decision and Control (CDC)*, 2023, pp. 8162–8169.
- [31] S. Wang, M. Diagne, and M. Krstic, “Backstepping neural operators for 2×2 hyperbolic PDEs,” *arXiv preprint arXiv:2303.10506*, 2023.
- [32] I. Wolfram Research, *Mathematica*. Champaign, IL: Wolfram Research, Inc., 2022.