MATHEMATICAL MODELING AND COMPUTING, Vol. 11, No. 2, pp. 448–454 (2024)

Physics-informed neural networks for the reaction-diffusion Brusselator model

Hariri I.[1], Radid A.[1], Rhofir K.[2]

[1]*LMFA, FSAC, Hassan II University of Casablanca*
[2]*LASTI, ENSAK, University of Sultan Moulay Slimane*

(Received 27 December 2023; Revised 16 May 2024; Accepted 18 May 2024)

In this work, we are interesting in solving the 1D and 2D nonlinear stiff reaction-diffusion Brusselator system using a machine learning technique called Physics-Informed Neural Networks (PINNs). PINN has been successful in a variety of science and engineering disciplines due to its ability of encoding physical laws, given by the PDE, into the neural network loss function in a way where the network must not only conform to the measurements, initial and boundary conditions, but also satisfy the governing equations. The utilization of PINN for Brusselator system is still in its infancy, with many questions to resolve. Performance of the framework is tested by solving some one and two dimensional problems with comparable numerical or analytical results. Validation of the results is investigated in terms of absolute error. The results showed that our PINN has well performed by producing a good accuracy on the given problems.

Keywords: *Physics-Informed Neural Network; deep learning; reaction-diffusion Brusselator system; stiff PDEs.*

2010 MSC: 68T07, 68Txx, 35K57 **DOI:** 10.23939/mmc2024.02.448

1. Introduction

The Brusselator model studied in this paper is used to describe an auto-catalytic chemical reaction between two reactant substances where one of them interacts with the other in order to increase its production rate [1]. Introduced by Prigogine Ilya [2], the reaction-diffusion Brusselator model is described by the following reactions:

$$\begin{gathered} A \xrightarrow{r_1} U, \\ B + U \xrightarrow{r_2} V + F, \\ 2U + V \xrightarrow{r_3} 3U, \\ U \xrightarrow{r_4} E, \end{gathered}$$

where the positive parameters r_i, $(i = 1, 2, 3, 4)$ represent the reaction rate constant. In the system above, a reactant, A, is transformed to a final product E in four steps with the help of four additional species U, B, V, and F. A and B can be modeled at constant concentrations because they are considered to be in vast excess.

Let $\Omega \subset \mathbb{R}^2$ and $\partial\Omega$ is its boundary. The Brusselator system is characterized by the following system of PDEs [3]:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \mu\Delta u + u^2 v - (\varepsilon + 1)u + \beta, \\ \dfrac{\partial v}{\partial t} = \mu\Delta v - u^2 v + \varepsilon u, \end{cases} \tag{1}$$

with the given Dirichlet and Neumann boundary conditions:

$$\begin{cases} u(x, y, t) = h_1(x, y, t), \quad (x, y) \in \partial\Omega, \\ v(x, y, t) = h_2(x, y, t), \\ u_x(x, y, t) = 0, \\ v_y(x, y, t) = 0, \end{cases}$$

and the following initial conditions:

$$\begin{cases} u_0(x,y) = s_1(x,y), & (x,y) \in \Omega, \\ v_0(x,y) = s_2(x,y). \end{cases}$$

In above equations, u and v are the dimensionless concentrations of the reactants U and V, β and ε are the constant concentrations of input reactants, μ is a constant diffusion coefficient, h_1, h_2, s_1 and s_2 are known functions. Δ is the Laplacian operator.

The reaction-diffusion Brusselator model has diverse applications in chemical reaction-diffusion processes such as enzymatic reactions, the formation of ozone by atomic oxygen via triple collision, multiple coupling between modes in laser physics, and plasma physics.

Many researchers used several numerical methodes to approximate the solutions due to the widespread use of the model and the inaccessibility of the exact solution. For example, in [4] Sirajul Haq et al. presented a numerical scheme based on Fibonacci and Lucas polynomials. In [5], the authors have used a combined cubic B-spline method with RK4 scheme for the resolution of two-dimensional system. In [6], the authors developed a local discontinuous Galerkin and variational multiscale element free Galerkin methods for the numerical solution of the two-dimensional Brusselator model. Twizell et al. [7] applied the second-order finite difference scheme in order to obtain numerical solution of the diffusion free Brusselator model.

In recent years, Deep Learning methods have been achieving unparalleled success in many application fields such as speech recognition [8], image recognition [9,10] and natural language processing [11]. In particular, Physics-Informed Neural Networks (PINNs) have been used to solve numerous forward and inverse differential problems. According to the results, PINNs can effectively solve partial differential equations at given certain initial and boundary conditions. Compared to traditional numerical methods, neural networks offer several advantages, including higher accuracy and more efficient computation. Many authors have used this approach in order to solve stiff PDEs such as the Brusselator system [12, 13].

In the current work, we will evaluate the PINN's performance in solving 1D and 2D Brusselator problems and compare it to those of other numerical methods [4, 5]. In Section 2, an explanation of the PINN methodology will be presented. In Section 3, we will study the performances of the PINN in solving four Brusselator problems. Conclusions are presented in Section 4.

2. PINN's methodology

Physics-Informed Neural Networks are neural networks (NN) that are trained to solve supervised learning problems while respecting the physical laws are given by the PDE. They were proposed by Raissi et al. [14].

PINNs can solve PDEs expressed in this general form [15]:

$$\begin{cases} \mathcal{N}(u(x);\lambda) = f(x) & x \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}(u(x)) = g(x) & x \in \partial\Omega, \end{cases}$$

where $\mathcal{N}$ represents the nonlinear differential operator, $\mathcal{B}$ is the initial/boundary conditions operator, $x := (x_1, \ldots, x_{d-1}, t)$ represents the spatiotemporal coordinate vector defined on Ω and λ are the parameters related to the physics. Here, u represents the unknown solution of the problem, f is the function that identify the data of the problem and g is the boundary function.

In the PINN methodology, the unknown solution $u(x)$ is computationally predicted by a fully connected neural network, parameterized by a set of parameters θ. This network takes the space-time coordinate vector x as input and then outputs a solution $\hat{u}_\theta(x)$. Between the input and the output layers exist multiple hidden layers. Each of these hidden layer takes $X = [x_1, x_2, \ldots, x_j]$ as an input and outputs $Y = [y_1, y_2, \ldots, y_i]$ through a nonlinear activation function $\sigma(\cdot)$ such as:

$$y_i = \sigma(w_{i,j}x_j + b_i)$$

where trainable hyperparameters $w_{i,j}$ and b_i represent the weights and biases of the network.

In this sense, the NN must learn to approximate the solution of the partial differential equation through finding the hyperparameters θ defining the network by minimizing a weighted loss function:

$$\theta^* = \arg\min_{\theta}(\mathcal{L}(\theta, \mathcal{T})).$$

The loss function is given by:

$$\mathcal{L}(\theta, \mathcal{T}) = \omega_{\mathcal{N}}\mathcal{L}_{\mathcal{N}}(\theta, \mathcal{T}_n) + \omega_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\theta, \mathcal{T}_b),$$

where:

$$\mathcal{L}_{\mathcal{N}}(\theta, \mathcal{T}_n) = \frac{1}{|\mathcal{T}_n|}\sum_{x\in\mathcal{T}_n} \|\mathcal{N}(\hat{u}_\theta(x); \lambda) - f(x)\|_2^2,$$

$$\mathcal{L}_{\mathcal{B}}(\theta, \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|}\sum_{x\in\mathcal{T}_b} \|\mathcal{B}(\hat{u}_\theta(x)) - g(x)\|_2^2.$$

Here, $\mathcal{L}_{\mathcal{N}}$ and $\mathcal{L}_{\mathcal{B}}$ are the residual of the governing PDE and the boundary conditions respectively. $\mathcal{T}_n$ is the set of points inside the domain Ω and $\mathcal{T}_b$ is the set of points on the boundary $\partial\Omega$. $\omega_{\mathcal{N}}$ and $\omega_{\mathcal{B}}$ are the weighting coefficients of $\mathcal{L}_{\mathcal{N}}$ and $\mathcal{L}_{\mathcal{B}}$ respectively.

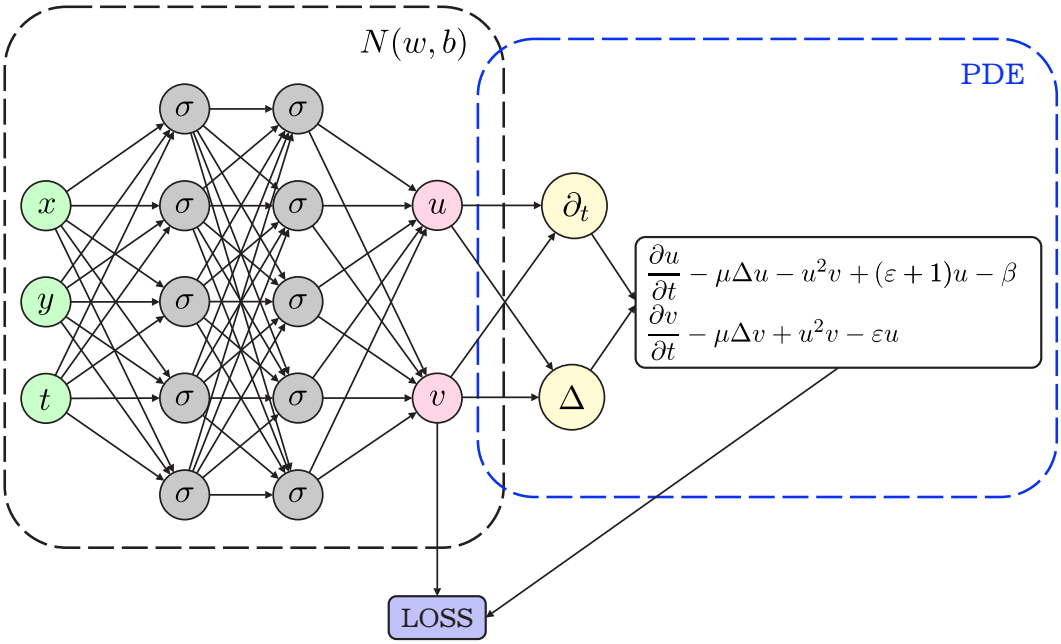

Fig. 1. Schematic of PINN for solving the Brusselator system.

In Figure 1, we present a schematic of our PINN framework. The NN takes as input the spatiotemporal coordinates x, y and t, and through its hidden layers it outputs the concentrations u and v. Then the network calculates the boundary losses from the boundary conditions and then uses the automatic differentiation to calculate the residual inside the domain by enforcing the PINN output to satisfy the Brusselator equations. The total loss is then calculated by adding the PDE residual loss and the boundary loss. The loss function is then minimized using an optimizer in order to obtain the new hyperparametres of the network.

3. Results and discussions

We test the efficiency of our PINN on four different problems. We implemented our PINN using Google Colaboratory notebooks `https://colab.research.google.com/?hl=fr` on its GPU T4. We sample the training points using the Latin Hypercube Sampling (LHS) and we choose the activation function for all the examples to be "tanh(·)".

Problem 1. We consider the case of one dimensional Brusselator model given by:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \mu \dfrac{\partial^2 u}{\partial x^2} + u^2 v - (\varepsilon + 1)u + \beta, \\ \dfrac{\partial v}{\partial t} = \mu \dfrac{\partial^2 v}{\partial x^2} - u^2 v + \varepsilon u, \end{cases}$$

with the following initial conditions:

$$u_0(x) = 1 + \sin(2\pi x), \quad v_0(x) = 3 \quad x \in \Omega,$$

and Dirichlet boundary conditions:

$$u(0,t) = 1, \quad u(1,t) = 1, \quad v(0,t) = 3, \quad v(1,t) = 3, \quad t > 0.$$

We choose, in this case, $\mu = 1/50$, $\varepsilon = 3$ and $\beta = 1$. We choose 30000 training residual points inside Ω, and 3000 training points sampled on its boundary for the boundary conditions and 3000 initial residual points for the initial conditions. We use a Fully Connected Neural Network (FCNN) of depth 5 (i.e., 4 hidden layers) with 50 neurons on each layer, we trained our NN with 30000 iterations using the optimizer Adam then we continue to train our neural network using the optimizer L-BFGS to achieve a smaller loss. The predicted solution was plotted using our PINN over the domain $\Omega = [0,1] \times [0,10]$. As shown in Figure 2, the solution profile is in good agreement with those computed using numerical methods in [4].

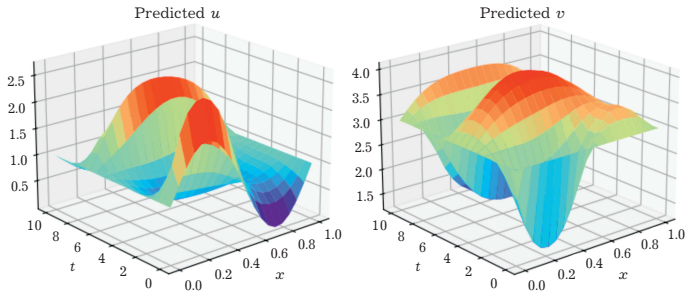

Fig. 2. Predicted solution for Problem 1.

Problem 2. For this problem, we consider equation (1) with the given initial conditions:

$$u_0(x,y) = 0.5 + y, \quad v_0(x,y) = 1 + 5x,$$

and boundary conditions:

$$u_x(0,y,t) = 0, \quad u_x(1,y,t) = 0, \quad v_y(x,0,t) = 0, \quad v_y(x,1,t) = 0, \quad t > 0.$$

We choose $\varepsilon = 0.5$, $\beta = 1$ and $\mu = 0.002$. We choose 30000 training residual points inside the domain $\Omega = [0,1]^2 \times [0,10]$, and 1500 training points sampled on its boundary. We also choose 2000 initial residual points for the initial conditions. We use a FCNN of depth 4 (i.e., 3 hidden layers) with 123 neurons on each layer, we trained our NN with 30000 iterations using Adam then we continue the training of our the network using the optimizer L-BFGS to achieve a smaller loss. In Table 1, we compare our PINN with the numerical method cited in [5]. From the table it is noticed that our method produced same results as recorded in [5].

Table 1. Approximate solution of Problem 2 at point $(0.4, 0.6)$.

T	Our PINN	Ref. [5]
1.0	$u = 2.3890$	$u = 2.3893$
	$v = 0.1965$	$v = 0.1968$
2.0	$u = 1.4464$	$u = 1.4478$
	$v = 0.2970$	$v = 0.2971$
3.0	$u = 1.1087$	$u = 1.1094$
	$v = 0.3877$	$v = 0.3877$
5.0	$u = 0.9796$	$u = 0.9799$
	$v = 0.4859$	$v = 0.4858$
7.0	$u = 0.9920$	$u = 0.9925$
	$v = 0.5038$	$v = 0.5035$
8.0	$u = 0.9977$	$u = 0.9979$
	$v = 0.5029$	$v = 0.5026$
9.0	$u = 1.0003$	$u = 1.0001$
	$v = 0.5014$	$v = 0.5011$

Problem 3. We consider another case of equation (1) when $\mu = 0.25$, $\varepsilon = 1$ and $\beta = 0$ with the given exact solution for all $(x,y) \in [0,1]^2$ and $t \in [0,5]$:

$$\begin{cases} u(x,y,t) = \exp(-x - y - 0.5t), \\ v(x,y,t) = \exp(x + y + 0.5t). \end{cases}$$

In this case, we choose 60000 points inside the domain and 6000 points for the boundary and initial conditions. We use the same architecture of NN as used in Problem 1. We trained our network for 30000 iterations using Adam then we continue training the network using L-BFGS to obtain a smaller loss. We presented the results obtained by our PINN in Tables 2 and

3. It can be seen from the tables that the proposed framework produced accurate solutions. Figures 3 and 4 show an implementation of the predicted solution at $T = 2$ and $T = 5$ respectively. From the figures we can notice that the exact and the predicted solutions are well matched with each other which prove that the proposed PINN is efficient.

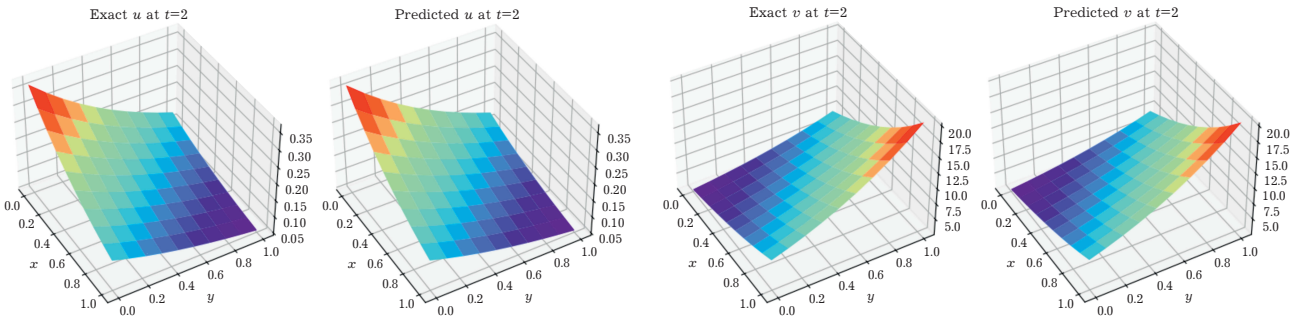

Fig. 3. Exact and predicted solutions at $t = 2$.

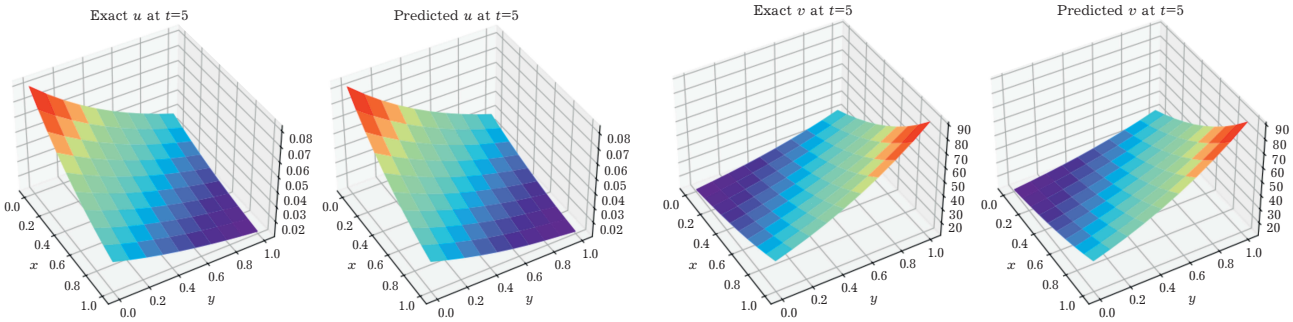

Fig. 4. Exact and predicted solutions at $t = 5$.

Table 2. Absolute errors for Problem 3 using PINN solution.

(x, y)	T	Exact u	Absolute u	Absolute error
$(0.3, 0.3)$	1	0.33281922	0.33287108	5.1856×10^{-5}
	3	0.12242617	0.12245643	3.0264×10^{-5}
	5	0.04563612	0.04504920	5.8692×10^{-4}
$(0.5, 0.5)$	1	0.22308332	0.22313016	4.4894×10^{-5}
	3	0.08185102	0.08208500	2.3380×10^{-4}
	5	0.03043715	0.03019738	2.3977×10^{-4}
$(0.9, 0.9)$	1	0.09998850	0.10025884	2.7034×10^{-4}
	3	0.03648921	0.03688317	3.9396×10^{-4}
	5	0.01383702	0.01356856	2.6846×10^{-4}

Table 3. Absolute errors for Problem 3 using PINN solution.

(x, y)	T	Exact v	Absolute v	Absolute error
$(0.3, 0.3)$	1	3.00361630	3.00416602	5.4979×10^{-4}
	3	8.16627900	8.16616991	1.0872×10^{-4}
	5	22.19785700	22.19795128	9.3460×10^{-5}
$(0.5, 0.5)$	1	4.4813876	4.48168907	3.0136×10^{-4}
	3	12.18246100	12.18249396	3.3379×10^{-5}
	5	33.11552000	33.11545196	6.8664×10^{-5}
$(0.9, 0.9)$	1	9.9736960	9.97418245	4.8637×10^{-4}
	3	27.11241100	27.11263892	2.2697×10^{-4}
	5	73.70000000	73.69979370	2.0599×10^{-4}

Problem 4. Consider equation (1) with the following initial conditions:

$$u_0(x,y) = 2 + 0.25y, \quad v_0(x,y) = 1 + 0.8x \quad (x,y) \in [0,1]^2,$$

and boundary conditions:

$$u_x(0,y,t) = 0, \quad u_x(1,y,t) = 0, \quad v_y(x,0,t) = 0, \quad v_y(x,1,t) = 0, \quad t \in [0,10].$$

We choose the parameters to be $\mu = 0.002$, $\varepsilon = 1$ and $\beta = 2$. We implemented the same NN used in Problem 3. We compare the results obtain with PINN with those obtained in [4, 5]. The results are presented in Tables 4 and 5. From the tables, we can see that our framework gives the same results as in [4, 5].

Table 4. Approximate solution of Problem 4 at point $(0.2, 0.2)$.

T	Our PINN	Ref. [4]	Ref. [5]
1.0	$u = 2.3453$	$u = 2.3457$	$u = 2.3454$
	$v = 0.4163$	$v = 0.4162$	$v = 0.4163$
2.0	$u = 2.0950$	$u = 2.0953$	$u = 2.0952$
	$v = 0.4690$	$v = 0.4688$	$v = 0.4689$
3.0	$u = 2.0218$	$u = 2.0220$	$u = 2.0219$
	$v = 0.4917$	$v = 0.4916$	$v = 0.4916$
5.0	$u = 2.0007$	$u = 2.0008$	$u = 2.0008$
	$v = 0.4997$	$v = 0.4997$	$v = 0.4996$
7.0	$u = 2.0000$	$u = 2.0000$	$u = 2.0001$
	$v = 0.5000$	$v = 0.5000$	$v = 0.4999$
8.0	$u = 2.0000$	$u = 2.0000$	$u = 2.0000$
	$v = 0.5000$	$v = 0.5000$	$v = 0.5000$
9.0	$u = 2.0001$	$u = 2.0000$	$u = 2.0000$
	$v = 0.5000$	$v = 0.5000$	$v = 0.5000$

Table 5. Approximate solution of Problem 4 at point $(0.8, 0.9)$.

T	Our PINN	Ref. [4]	Ref. [5]
1.0	$u = 2.6073$	$u = 2.6323$	$u = 2.6069$
	$v = 0.3692$	$v = 0.3656$	$v = 0.3693$
2.0	$u = 2.1759$	$u = 2.1839$	$u = 2.1757$
	$v = 0.4476$	$v = 0.4459$	$v = 0.4478$
3.0	$u = 2.0428$	$u = 2.0460$	$u = 2.0433$
	$v = 0.4845$	$v = 0.4837$	$v = 0.4844$
5.0	$u = 2.0015$	$u = 2.0027$	$u = 2.0017$
	$v = 0.4993$	$v = 0.4991$	$v = 0.4993$
7.0	$u = 1.9999$	$u = 2.0011$	$u = 2.0001$
	$v = 0.5000$	$v = 0.4997$	$v = 0.4999$
8.0	$u = 2.0000$	$u = 2.0011$	$u = 2.0000$
	$v = 0.5000$	$v = 0.4998$	$v = 0.5000$
9.0	$u = 2.0000$	$u = 2.0000$	$u = 2.0000$
	$v = 0.5000$	$v = 0.5000$	$v = 0.5000$

4. Conclusions

In this study, we test the efficiency of PINNs in solving the nonlinear reaction diffusion Brusselator system. A total of four problems of the Brusselator system were investigated using the proposed technique.

We compared the results obtained by PINN with those obtained by the numerical methods cited in [4, 5]. It was found that the solutions obtained by PINN are very accurate and show good agreement with the numerical and analytical solutions.

For future work, we will compare the performance of PINN with other neural network techniques, such as Recurrent Neural Networks (RNN).

[1] Ahmed N., Rafiq M., Rehman M. A., Iqbal M. S., Ali M. Numerical modeling of three dimensional Brusselator reaction diffusion system. AIP Advances. **9** (1), 015205 (2019).

[2] Prigogine I. Time, structure, and fluctuations. Science. **201** (4358), 777–785 (1978).

[3] Adomian G. The diffusion-Brusselator equation. Computers & Mathematics with Applications. **29** (5), 1–3 (1995).

[4] Haq S., Ali I., Nisar K. S. A computational study of two-dimensional reaction–diffusion Brusselator system with applications in chemical processes. Alexandria Engineering Journal. **60** (5), 4381–4392 (2021).

[5] Jiwari R., Yuan J. Computational modeling of two dimensional reaction–diffusion Brusselator system arising in chemical processes. Journal of Mathematical Chemistry. **52**, 1535–1551 (2014).

[6] Dehghan M., Abbaszadeh M. Variational multiscale element free Galerkin (VMEFG) and local discontinuous Galerkin (LDG) methods for solving two-dimensional Brusselator reaction–diffusion system with and without cross-diffusion. Computer Methods in Applied Mechanics and Engineering. **300**, 770–797 (2016).

[7] Twizell E. H., Gumel A. B., Cao Q. A second-order scheme for the "Brusselator" reaction–diffusion system. Journal of Mathematical Chemistry. **26**, 297–316 (1999).

[8] Zhang S., Chen M., Chen J., Li Y.-F., Wu Y., Li M., Zhu C. Combining cross-modal knowledge transfer and semi-supervised learning for speech emotion recognition. Knowledge-Based Systems. **229**, 107340 (2021).

[9] Gao Y., Mosalam K. M. Deep Transfer Learning for Image-Based Structural Damage Recognition. Computer-Aided Civil and Infrastructure Engineering. **33** (9), 748–768 (2018).

[10] Yang X., Zhang Y., Lv W., Wang D. Image recognition of wind turbine blade damage based on a deep learning model with transfer learning and an ensemble learning classifier. Renewable Energy. **163**, 386–397 (2021).

[11] Ruder S., Peters M. E., Swayamdipta S., Wolf T. Transfer Learning in Natural Language Processing. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials. 15–18 (2019).

[12] Kontolati K., Loukrezis D., Giovanis D. G., Vandanapu L., Shields M. D. A survey of unsupervised learning methods for high-dimensional uncertainty quantification in black-box-type problems. Journal of Computational Physics. **464**, 111313 (2022).

[13] Anantharaman R., Abdelrehim A., Jain A., Pal A., Sharp D., Utkarsh, Edelman A., Rackauckas C. Stably Accelerating Stiff Quantitative Systems Pharmacology Models: Continuous-Time Echo State Networks as Implicit Machine Learning. IFAC-PapersOnLine. **55** (23), 1–6 (2022).

[14] Raissi M., Perdikaris P., Karniadakis G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. **378**, 686–707 (2019).

[15] Cuomo S., Di Cola V. S., Giampaolo F., Rozza G., Raissi M., Piccialli F. Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next. Journal of Scientific Computing. **92**, 88 (2022).

Фізичні нейронні мережі для реакційно-дифузійної моделі брюселятора

Харіpі І.[1], Радід А.[1], Рофір К.[2]

[1]*LMFA, FSAC, Університет Хасана II Касабланки*
[2]*LASTI, ENSAK, Університет Султана Мулая Слімана*

У цій роботі розв'язуємо одновимірну та двовимірну нелінійну жорстку реакційно-дифузійну систему брюселятора за допомогою техніки машинного навчання під назвою фізичні нейронні мережі (PINN). PINN досяг успіху в різних наукових та інженерних дисциплінах завдяки своїй здатності кодувати фізичні закони, які задані диференціальними рівняннями у частинних похідних, у функцію втрат нейронної мережі так, що мережа повинна не лише відповідати вимірюванням, початковим і граничним умовам, але також задовольняти основні рівняння. Використання PINN для брюселятора все ще перебуває в зародковому стані, і потрібно вирішити багато питань. Ефективність фреймворку перевіряється шляхом розв'язання деяких одно- та двовимірних задач із порівнянням їх з чисельними або аналітичними результатами. Перевірка результатів досліджується з точки зору абсолютної похибки. Результати показали, що наш PINN спрацював добре, забезпечивши високу точність у розв'язанні поставлених задач.

Ключові слова: *фізична нейронна мережа; глибоке навчання; реакційно-дифузійна система Брюсселятора; жорсткі PDE.*