Exploring Datasets to Solve Partial Differential Equations with TensorFlow

Oscar G. Borzdynski[1], Florentino Borondo[2,3(✉)], and Jezabel Curbelo[1,2]

[1] Departamento de Matemáticas, Universidad Autónoma de Madrid, Cantoblanco, 28049 Madrid, Spain
[2] Instituto de Ciencias Matemáticas (ICMAT), Cantoblanco, 28049 Madrid, Spain
[3] Departamento de Química, Universidad Autónoma de Madrid, Cantoblanco, 28049 Madrid, Spain
f.borondo@uam.es

Abstract. This paper proposes a way of approximating the solution of partial differential equations (PDE) using Deep Neural Networks (DNN) based on Keras and TensorFlow, that is capable of running on a conventional laptop, which is relatively fast for different network architectures. We analyze the performance of our method using a well known PDE, the heat equation with Dirichlet boundary conditions for a non-derivable non-continuous initial function. We have tried the use of different families of functions as training datasets as well as different time spreadings aiming at the best possible performance. The code is easily modifiable and can be adapted to solve PDE problems in more complex scenarios by changing the activation functions of the different layers.

Keywords: Deep learning · Partial derivative equations · TensorFlow · Keras · Neural Network

1 Introduction

The use of Machine Learning (ML) is spreading across many fields in Applied Science, often showing a very good performance in the resolution of many different practical tasks, such as weather forecasting [14], self driving cars [12], or translation [2], just to name a few. However, ML is not very popular in Mathematics or other theoretical sciences, despite the fact that strong evidence of its great potential has been recently reported in the literature [6]. Reservoir computing [11], for example, is one such method, which unfortunately is very demanding computationally.

In this paper we explore a more economic computationally alternative way of approximating the numerical solution of Partial Differential Equations using Deep Neural Networks (DNN) based on the Keras [4] and Tensorflow softwares [1]. This framework is widely used for its performance and versatility [5].

Á. Herrero et al. (Eds.): SOCO 2020, AISC 1268, pp. 441–450, 2021.
https://doi.org/10.1007/978-3-030-57802-2_42

Table 1. DNN Structure and activation functions

Layer	No. of neurons	Activation function
Entry layer	100	Linear
First hidden layer	1250	Linear
Second hidden layer	2500	Linear
Third hidden layer	5000	Linear
Exit layer	5000	Linear

Deep learning techniques are promising in solving PDEs because they are able to represent complex-shaped functions very effectively, specially when compared to other traditional methods which experience the "curse of dimensionality" difficulties. For instance, the experiments in [8] show that the artificial neural networks exhibit a better performance than finite element methods for several cases of PDEs.

Similar work to ours has been reported in the literature. In particular, DeepXDE [9] is a code made to solve PDE using Tensorflow that allows the user to make an approximation without making a big effort in choosing the structure of the DNN. Good results have been obtained using this library. For example, it has been applied to the study of inverse problems in nano-optics and metamaterials [3], and space-time fractional advection-diffusion equations [10]. We decided to use plain Tensorflow to be capable of finely tuning the network for our problem.

To illustrate and analyze the feasibility and performance of our method we apply it to a well known PDE, as it is the heat equation [15] with Dirichlet boundary conditions for a non-derivable non-continuous initial function. We tried different families of particular solutions as training datasets, and check the use of different ways to span the time interval, seeking for the best performance. Excellent solutions are found for generic initial functions in all cases explored so far.

The paper is organized in four sections. In Sect. 2, we describe the Deep Neural Network structure, activation functions, and training dataset. Sect. 3 is devoted to briefly explain the heat equation and its possible theoretical solutions. In Sect. 4, we illustrate our method by presenting the results obtained in several numerical experiments. Finally, in Sect. 5 we summarized our conclusions, and discuss possibilities for future work.

2 Data and Methods

2.1 Our Deep Neural Network

A DNN is formed by a series of layers, each one consisting a certain number of neurons with a given activation function. The activation function defines the information flows along the network.

The parameters defining the structure of our DNN are given in Table 1. The entry layer receives 100 equidistant samples of the initial function. The hidden layers are incremental with 1250, 2500 and 5000 neurons, respectively. The exit layer has 5000 outputs which correspond to a matrix of 100×50 with the first dimension being the position and the second the time.

This structure is chosen for several reasons. The first one is the possibility of predicting non-bounded negative values, the linear activation function makes this possible since it is defined in the range $(-\infty, \infty)$. We consider that the neurons receive a vector X and they have a vector of weights W where W_i corresponds to the input X_i referring i neurons in the previous layer. A linear activation function means that the exit signal of the neurons is $W^\intercal X$, this implying a linear transformation of the input to the output data. Second, the behavior of the activation function near zero is not as steep as others functions, like for example the sigmoid [6]. The third is the growing effect obtained from an increasing number of neurons, adding information instead of removing or shuffling it.

As the last parameters of out network we need to specify an optimizer and a loss function. The loss function is the objective to minimize, it compares the exit of the DNN with the expected result, and returns a metric which indicates the distance between them. The optimizer is the algorithm that determines how the parameters of the network change to minimize the loss, fitting the data to the expected result.

We decided to use the root mean square error (RMSE) as loss function because it penalizes big errors, and we want a uniform fit to the solution. Also we used the well known ADAM optimizer [7] as it has been empirically shown [13] to work well, improving the performance of other alternative methods.

2.2 Our Training Datasets

The dataset that we use consists of 2000 equations, and we train our DNN with 1600 (80%) of them in batches of 100. After several tries we decided that 20 epochs were sufficient to achieve acceptable results.

3 Example: The Heat Equation

To illustrate the performance of our method we use the well known heat equation with Dirichlet boundary conditions

$$\begin{cases} u_t = \alpha u_{xx} & t > 0, 0 < x < L, \\ u(x,0) = f(x) & 0 < x < L, \\ u(0,t) = u(L,t) = 0 & t > 0, \end{cases} \tag{1}$$

which solution is

$$\begin{aligned} u(x,t) &= \sum_{n=1}^{\infty} \left\{ b_n \sin \frac{nx\pi}{L} \exp\left[-\left(\frac{n\alpha\pi}{L}\right)^2 t \right] \right\}, \\ b_n &= \frac{2}{L} \int_0^L f(x) \sin \frac{nx\pi}{L} dx. \end{aligned} \tag{2}$$

Table 2. Definition of datasets and testing ways in the different experiments. Linear and exponential time means, respectively, that the time steps are equally, or exponentially, separated times (see text for details).

Experiment	Training dataset	Testing
A	Family of functions $f(x)$ defined in Eq. (4) with different random intervals and linear time	$h(x)$ defined in Eq. (5) and linear time
B	Family of functions $f(x)$ defined in Eq. (3) with different random a parameter and linear time	$h(x)$ defined in Eq. (5) and linear time
C	Family of functions $f(x)$ defined in Eq. (3) with different random a parameter and exponential time	$h(x)$ defined in Eq. (5) with exponential time
D	Family of functions $f(x)$ defined in Eq. (3) with different random a parameter and linear time	$h(x)$ defined in Eq. (5) with linear time and double interval

assuming a one-dimensional rod of length $L = \pi$, $0 \leq t < 0.05$, and defining two different initial conditions:

$$f(x) = \sin(ax), \tag{3}$$

$$f(x) = \begin{cases} 1 \text{ if } x \in I \text{ and,} \\ 0 \text{ otherwise,} \end{cases} \tag{4}$$

where a is a characteristics parameter, and I is a given known set consisting of two non-overlapping intervals defined by four limits.

4 Results

In order to train the DNN a dataset is needed. We are going to explore four different ways of generating it, and one way of testing it, as summarized in Table 2.

In the first experiment A, the initial data $f(x)$ for Eq. (1) is given by (4) defined with random intervals I. For the second experiment B, $f(x)$ is given by (3) with random a. In both experiments the temporal grid is uniform, i.e. $t_i = 0.001i$ with $i = 0, \ldots, N-1$ where N is the number of temporal nodes. Experiment C is equal to experiment B but with the node distance following the expression $t_i = [-1 + \exp(i/N)]/20$ for $i = 0, \ldots, N-1$. Experiment D is the same as experiment B but with an extended (doubled) time interval.

In all the previous scenarios we use a test function $h(x)$, as initial condition of (1), which is the non-derivable non-continuous function:

$$h(x) = \begin{cases} 0 & \text{if} \quad x = 0 \\ 0.3 & \text{if} \quad 0 < x < \pi/2 \\ 0.8 & \text{if} \quad \pi/2 \leq x < \pi \\ 0 & \text{if} \quad x = \pi. \end{cases} \tag{5}$$

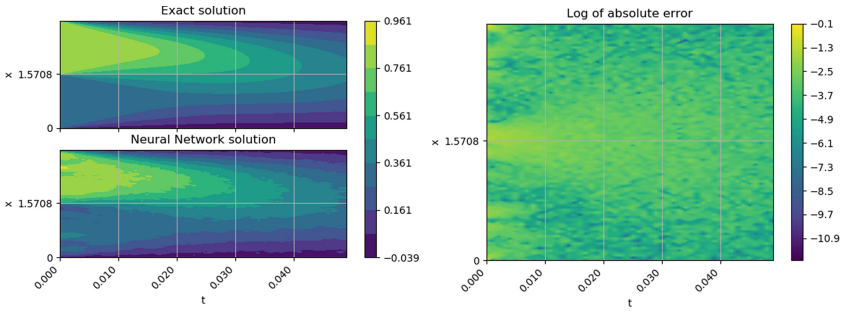

Fig. 1. Result of experiment A of Table 2. (Left) Theoretical and approximate solution obtained by the DNN. (Right) Logarithm of the error of the DNN approximation, where yellow/blue color means bigger/smaller errors. The maximum error occurs in the extremes of the rod at $t = 0$, where the model does not comply with the Dirichlet boundary condition. Its value is 0.8181 (top left corner). The maximun error for $t > 0$ is 0.2813 and the mean error is 0.0305.

Note that this function barely complies with the Dirichlet condition, and the solution is not easily computed as it need to be transformed to a Fourier series. Although $h(x)$ plays the same role as $f(x)$ in Eq. (1), we use different notation to easily differentiate the functions used for training and testing.

The hardware used in all the experiments is a very modest:

- i7-4790 8 threads 3.6 Ghz
- 16 GB of RAM
- 250 GB SSD

No use of the GPU (graphics processing card) was made, as it is customarily done, to test if a conventional computer was able to be trained and used to predict in a model like our. The typical time needed to generate the dataset was roughly 3 hours, and the training was performed in about 15 min.

In Figs. 1, 2, 3 and 4 we present the results obtained with the DNN specified in Table 1 for the four different scenarios described in Table 2.

In the first experiment A, we used a Dirac-delta shaped functions in random intervals. For testing, we used the function defined in Eq. (5). As can be seen in Fig. 1, the shape of the predicted and exact solutions are very similar, and the error is very uniform everywhere. The maximum error is 0.8181, which happens at the extreme of the rod, where the initial function has a very big gap. We will see that this effect happens in every test we have made. The mean error is 0.0305.

In the second experiment B we used a sin family of functions, while for testing the function defined in Eq. (5) is used. As it is seen in Fig. 2 errors mostly occurs at the initial time. We think that this is probably due to the big variation of $\sin(ax)$ when a is big. The maximum error takes a similar value as in experiment A, being this equal to 0.8134. The mean error is also similar to the previous case, and equal to 0.0330.

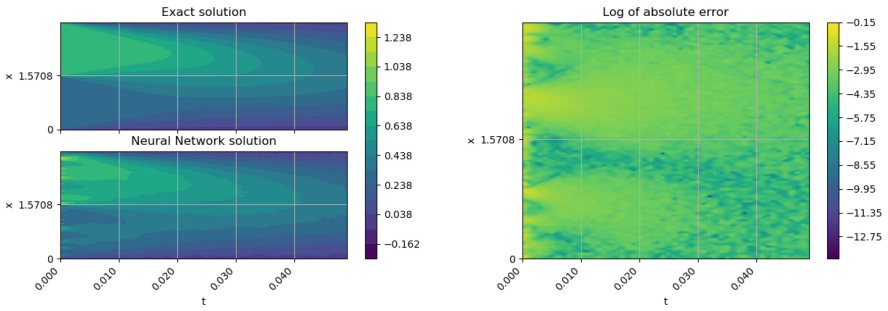

Fig. 2. Same as Fig. 1 for experiment B of Table 2. The corresponding maximums and mean error values are 0.8134, 0.3068, and 0.0330, respectively

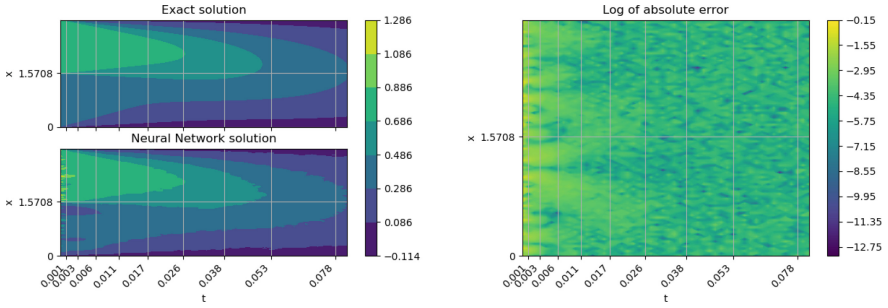

Fig. 3. Same as Fig. 1 for experiment C of Table 2. The corresponding maximums and mean error values are 0.8024, 0.2070, and 0.0179, respectively

In order to optimize the results obtained in the initial time portion, we bring closer the initial time steps and separate them a bit the further ones in experiment C (see results in Fig. 3). First, we see a big improvement in the mean error, being it reduced to 0.0179. The error at the end of the rod and the initial time is still the maximum obtained error, equal to 0.8024. We appreciate a smaller and more uniform error as time advances.

Finally, in the last experiment D, which results are shown in Fig. 4 we tried a new approach to see how the model works when the total time interval is extended (to twice the value used before in experiment C). To achieve this task, we reevaluated the last value obtained by the previous evaluation, obtaining twice the time. We see that the shape is similar, but the error gets a lot bigger, rising to 0.0506.

In all the experiments we have monitored the maximum error without considering the initial error at $t = 0$. The conclusion is that the error drastically decreases in all cases, but the maximum still happens as time goes to 0. Another conclusion that can be drawn from the previous results is that after those initial experiments, the best possible strategy is to stick to the exponential time approximation, since it renders the best results.

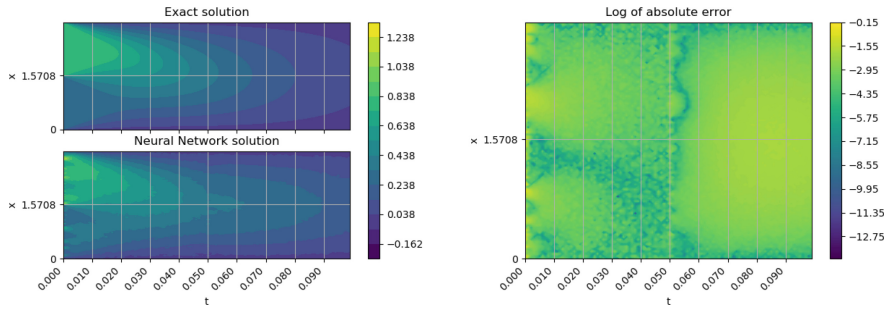

Fig. 4. Same as Fig. 1 for experiment D of Table 2. The corresponding maximums and mean error values are 0.8134, 0.3068, and 0.0506, respectively

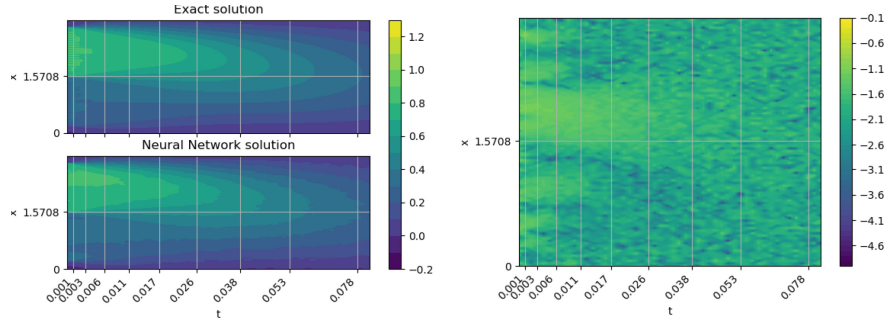

Fig. 5. (Left) Theoretical and approximate solution obtained by the DNN with exponential time for the function $\tilde{h}(x)$ defined in Eq. (6). (Right) Logarithm of the error of the DNN approximation, where yellow/blue color means bigger/smaller errors.

Therefore, we next try to evaluate smoother initial functions. Note that the used training dataset will be the same as before. In this new batch of numerical experiments, we try a function that is equal to $h(x)$ but not ending so close to the end of the rod, thus preventing the error at $t = 0$. For this purpose we use the following definition:

$$\tilde{h}(x) = \begin{cases} 0 & \text{if} \quad x \leq 0.2 \\ 0.3 & \text{if} \quad 0.2 < x < \pi/2 \\ 0.8 & \text{if} \quad \pi/2 \leq x < \pi - 0.2 \\ 0 & \text{if} \quad \pi - 0.2 \leq x. \end{cases} \tag{6}$$

The corresponding results are shown in Fig. 5, where we present the solution and the approximation made by our algorithm. Note that in this case the maximum error reduces to 0.0915, and that the mean error reduces to 0.0132. Also, it can be seen that the maximum error does not happen at $t = 0$. This result indicates that when the function does not fit the Dirichlet condition our method can not obtain a good approximation.

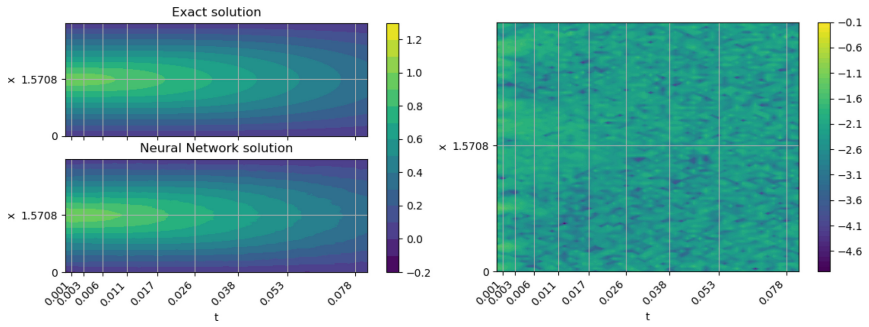

Fig. 6. Same as Fig. 5 for the function $\overline{h}(x)$ defined in Eq. (7).

We next try a new testing function that it is continuous, but non-derivable, defined in the following way:

$$\overline{h}(x) = \begin{cases} \dfrac{2x}{\pi} & \text{if} \quad x \leq \dfrac{\pi}{2}, \\ 2 - \dfrac{2x}{\pi} & \text{if} \quad x > \dfrac{\pi}{2}. \end{cases} \tag{7}$$

The corresponding results are presented in Fig. 6, where it is seen that the error is much smaller than in previous experiments; the maximum error is 0.0447, and the mean error 0.0055. Notice that this is the best case obtained along all our work, so that one can conclude that the use of continuous testing functions greatly improve the performance of our method.

As the last testing function, we try a continuous and derivable initial condition:

$$h^*(x) = -(x - 0)(x - \pi). \tag{8}$$

The corresponding results are presented in Fig. 7, where it is seen that a maximum error of 0.1177 and a mean error of 0.0156 are obtained. Notice that

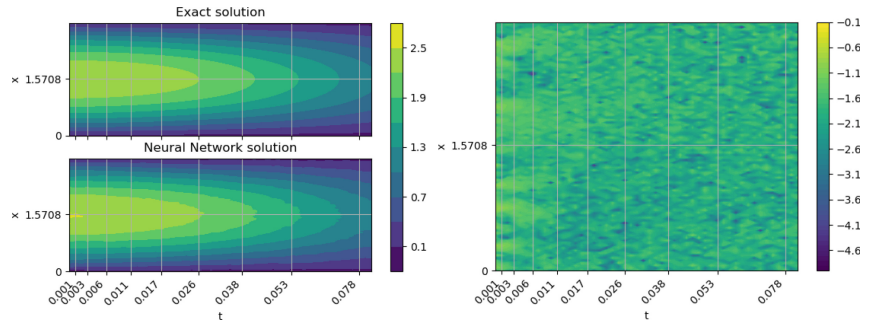

Fig. 7. Same as Fig. 5 for the function $h^*(x)$ defined in Eq. (8).

the maximum value of the solution is much larger than in the previous cases, so the percentage of error is more or less the same here.

5 Summary and Conclusions

In this work we have developed a simple DNN based on readily available software which is able to find accurate approximate numerical PDEs on modest laptop computers. We have use the well know heat equation to check the performance of the method. This represents a good alternative in terms of computational effort and cost to more sophisticated methods, such as the increasingly popular Reservoir Computing [11], whenever an extremely high accuracy is not required.

To optimize our DNN we have tried four different approaches, two families of functions and three different time spans, having obtained better results when compressing the time steps in the initial time and expanding them as time increases. Other initial functions where tested with this method, as the function smoothed the results improving them, achieving in this way a mean error of 10^{-3} in the best case.

The numerical experiments done in this paper show that deep learning may be used to approximate non-easily computable functions with a decent error in an everyday computer, even when the initial function does not fully comply with the boundary conditions. The only small problem of our approach is the generation of the training dataset, since a large number of solutions need to be computed. When the problem is theoretically solvable the required datasets can be easily obtained.

The main objective of our work was to develop a method able to run in an modest computer, thus making Deep Learning available to any researcher in computer science. Running it with a bigger dataset or more complex network structures will need bigger computational means that would improve the performance of our approximation. Also a combination of various families of functions has proven to improve the results, but we wanted to keep the dataset in this first paper as simple as possible.

Acknowledgments. This work has been partially supported by the Spanish Ministry of Science, Innovation and Universities, Gobierno de España, under Contracts No. PGC2018-093854-BI00, and ICMAT Severo Ochoa SEV-2015-0554, and from the People Programme (Marie Curie Actions) of the European Union's Horizon 2020 Research and Innovation Program under Grant No. 734557.

References

1. Abadi, M., et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015)
2. Bahdanau, D., Cho, K.H., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. International Conference on Learning Representations, ICLR (2015)

3. Chen, Y., Lu, L., Karniadakis, G.E., Dai Negro, L.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials, December 2019
4. Chollet, F., et al.: Keras (2015). https://github.com/fchollet/keras
5. Gulli, A., Pal, S.: Deep learning with Keras (2017)
6. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. National Acad. Sci. (USA) **115**(34), 8505–8510 (2018)
7. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. International Conference on Learning Representations, ICLR (2015)
8. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998)
9. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: a deep learning library for solving differential equations, July 2019
10. Pang, G., Lu, L., Karniadakis, G.E.: FPINNs: fractional physics-informed neural networks. SIAM J. Sci. Comput. **41**(4), A2603–A2626 (2019)
11. Pathak, J., Hunt, B., Girvan, M., Lu, Z., Ott, E.: Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. Phys. Rev. Lett. **120**(2), 1 (2018)
12. Ramos, S., Gehrig, S., Pinggera, P., Franke, U., Rother, C.: Detecting unexpected obstacles for self-driving cars: fusing deep learning and geometric modeling. In: IEEE Intelligent Vehicles Symposium, Proceedings, pp. 1025–1032. Institute of Electrical and Electronics Engineers Inc., July 2017
13. Ruder, S.: An overview of gradient descent optimization algorithms. ArXiv e-prints. https://arxiv.org/abs/1609.04747 (2016)
14. Salman, A.G., Kanigoro, B., Heryadi, Y.: Weather forecasting using deep learning techniques. In: ICACSIS 2015 - 2015 International Conference on Advanced Computer Science and Information Systems, Proceedings, pp. 281–285. Institute of Electrical and Electronics Engineers Inc., February 2016
15. Salsa, S.: A Primer on PDEs : Models, Methods, Simulations. La Matematica per il 3+2, 1st edn. (2013)