



HAL
open science

Generic method for solving partial differential equations through the design of problem-specific Cellular Neural Networks

Nicolas Fressengeas, Hervé Frezza-Buet

► **To cite this version:**

Nicolas Fressengeas, Hervé Frezza-Buet. Generic method for solving partial differential equations through the design of problem-specific Cellular Neural Networks. 2007. hal-00107064v5

HAL Id: hal-00107064

<https://hal.science/hal-00107064v5>

Preprint submitted on 24 May 2007 (v5), last revised 29 Jan 2010 (v8)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Generic method for solving partial differential equations through the design of problem-specific Cellular Neural Networks

N.Fressengeas

*Laboratoire Matériaux Optiques, Photonique et Systèmes, 57070 Metz, France
Unité de Recherche Commune à l'Université Paul Verlaine – Metz et Supélec - CNRS UMR 7132*

H.Frezza-Buet

Information, Multimodality and Signal, Supélec, 2, rue Edouard Belin, 57070 Metz, France

This paper presents an original and generic numerical method for solving partial differential equations. A new mathematical and systematic method stemming from the local very nature of any differential problem is proposed: a custom tailored Cellular Neural Network is purposely derived from any given differential problem so that its fixed point yields the solution in a quantitatively correct way. The combined use of formal computing and continuous automata thus offers the unique possibility to completely automate the process from formal problem specification to its numerical solution, thus potentially saving code development time for the numerical solving of partial differential problems.

PACS numbers: 02.30.Jr, 02.60.Cb, 02.60.Jh, 02.60.Lj

I. PARTIAL DIFFERENTIAL EQUATIONS AND CELLULAR NEURAL NETWORKS

A. Automata and calculus

Ever since von Neumann [1], the question of modelling continuous physics with a discrete set of cellular automata has been raised, whether they handle discrete or continuous values. Many answers have been brought forth through, for instance, the work of Stephen Wolfram [2] summarized in a recent book [3]. This problem has been mostly tackled by rightfully considering that modelling physics through Newton and Leibniz calculus is fundamentally different from a discrete modelization as implied by automata.

Indeed, the former implies that physics is considered continuous either because materials and fields are considered continuous in classical physics or because quantum physics wave functions are themselves continuous. On the contrary, modelling physics through automata implies modelling on a discrete basis, in which a unit element called *a cell*, interacts with its surroundings according to a given law derived from local physics considerations.

Such discretized automaton based models have been successfully applied to various applications ranging from reaction-diffusion systems [4] to forest fires [5], through probably one of the most impressive achievements: the Lattice Gas Automata [6], where atoms or molecules are considered individually. In this frame, simple point mechanics interaction rules lead to complex behaviors such as phase transition and turbulence. This peculiar feature of automata, making complex group behavior emerge from fairly simple individual rules aroused the interest around them for the past decades.

B. Cellular Neural Networks

Cellular automata-based modelling attempts have also concerned the theory of circuits for a few decades, because the Very-Large-Scale Integration (VLSI) components offer a large amount of configurable processors, spatially organized as a locally connected array of analogical and numerical processing units. In this field, the concept of Cellular Neural Network (CNN) has been proposed [7], which extends cellular automata, allowing local cells, that are dynamical systems, to deal with several continuous values and local connections.

CNNs are good candidates for the numerical resolution of partial differential equations (PDE), and a methodology for the design of a CNN from a given PDE has been proposed in [8]. This approach consists in performing a spatial discretization of the PDE through the finite difference scheme, yielding an Ordinary Differential Equation (ODE) on time that can be numerically solved by standard methods like Runge-Kutta.

This approach is widely used in this field, and drives the design of simulators like SCNN 2000 [9], as well as the design of actual VLSI components [10]. The partial differential system is there implemented using analogic VLSI components, the circuit temporal evolution being then the temporal evolution of the initial PDE.

Two main difficulties arise in this framework. The first one concerns the stability of the CNN dynamical system. Some stability studies of CNNs for classical PDEs can be found in [11] but stability has still to be analyzed when dealing with new specific problems, as it has been done, for example, for the dynamics of nuclear reactors [12]. The second difficulty raised by transforming PDE to ODE for resolution by CNNs is the actual fitting of the CNN to the PDE, since the method is more a heuristic one than a formal derivation of the CNN from the PDE, as mentioned in [13]. Furthermore, the features

of the CNN cannot be easily associated to the physical parameters involved in the PDE.

To cope with the lack of methods to formally derive a CNN from a PDE, some parameter tunings can be performed once the CNN is designed. This tuning can be driven by a supervised learning process, as in [9, 13]. Some other a posteriori checks can be achieved if some analytical solution of the PDE is known for particular cases, as in [11], or if some behavior of the CNN can be expected, as travelling waves or solitons [8, 14, 15]. In the latter case, validation is based on a qualitative criterion.

Some other methods to derive automata from particular differential problems such as reaction-diffusion systems [4] or Maxwell's equations [16] have been presented. In the former, the automaton is constructed from a moving average paradigm, while the latter is a modified version of the Lattice Gas Automaton [6].

C. Automata as quantitatively exact PDE solvers

In most cases, the predictions of calculus based, continuous models and those of discrete, automata based ones, are seldom quantitatively identical, though qualitative similarity is often obtained. This is mostly explained by the fact that the two drastically different approaches are applied to their own class of problems.

Some attempts have recently been made to set up the solution of a PDE by using a regression method [17]. The idea there is to measure an error at each discrete point of the system, and to drive an optimisation process in order to find the function that minimises this error, this function being taken in a parametrized set of continuous functions defined by a multi-layer perceptron. This error is null if the function that is found meets the EDP requirements. Such regression processes, based on classical empirical risk minimisation, are known to be sensitive to overfitting [18]. The idea of error minimisation in EDP requirements is also used in our approach, but we will show that it rather leads to the definition of a relaxation process that reaches a fixed point. This is completely different from searching a functional hypothesis space as in supervised learning techniques, and thus avoids all the generalization problems encountered in this field [18].

Other attempts at a quantitative link have however been made by showing connexions between an automaton and a particular differential problem [19] or by designing methods for describing automata by differential equations [20, 21, 22] allowing in the way to *assess* the performance of two different *implementations* of the same problem, which are in fact basically two different automata for the description of the same physics.

The interest of solving PDEs with cellular automata is of course not limited to physics, since PDEs are also intensively used in image processing [23], and some CNN-based solutions have also been proposed in that field [24]. This stresses the need for generic tools for simulating PDEs in many areas. In [24], an attempt has been made

to provide ready-to-use programming templates for the design of CNNs, and previously mentioned softwares [9] help to rationalize the design of CNNs for PDEs.

This paper is devoted to the introduction of a recent systematic method allowing to derive a cellular neural network from any given differential problem whose boundary conditions are of the Dirichlet type. It is an attempt to bridge the present gap [13] between continuous PDE and discrete CNN.

The process of derivation stems from the idea that since differential problems are expressed in a purely local manner, their solution can be computed in an equally local way. However, as will be shown, the locality of the computation is not an *a priori* hypothesis but rather a consequence of the method, which is essentially based on a relaxation process which implements the multidimensional Newton minimization method, thus ensuring stability.

Furthermore, the most interesting aspect of it is the possibility to completely automate the way from the formal expression of the differential problem down to its solution, thanks to formal computing and to a cellular neural network based environment, analogous to previously reported ones [25, 26]. This would indeed allow to save code development time for the numerical solving of a given partial differential problem, and would also reduce the computer programming skills required for this task.

II. FROM PDE TO CELLULAR NEURAL NETWORK

In this section, we will show that the solution seeking scheme of finding a field that meets the requirements of a given differential problem can be transformed into a minimization task. The latter can be implemented by formally computing a cellular automaton which converges to the sought solution.

For this demonstration to be precise, a mathematical formalism which can at first seem quite abstract, has to be used. To overcome this difficulty, let us particularize the formalism, at each step, to a simple example: the monodimensional Poisson equation, $\Delta V(x) = \frac{\partial^2 V}{\partial x^2} = \rho(x)$, V being the unknown electrostatic potential and ρ a given repartition of charges. The example chosen has of course a straightforward solution but it is simple enough so that each step can be detailed in the paper.

A. Definitions

The very characteristic of continuous physics is its intensive use of *fields*. If we note $(A)^B$ the set of functions from A to B , a field $\xi \in (\mathbb{R}^m)^{\mathbb{R}^n}$ is a mapping of a given vectorial physical quantity —belonging to \mathbb{R}^n — over a given physical space \mathbb{R}^m , for $(m, n) \in \mathbb{N}^2$. For instance, our example electrostatic potential field in a 1D space is a

scalar mapping over \mathbb{R} , as an electric field over a 3D space would be a 3D vector mapping over \mathbb{R}^3 . Furthermore, if time were present in this example, it would be treated equally as just an additional dimension. For instance, a time-resolved 3 dimensional problem is considered as having 4 dimensions.

Therefore, a particular local differential problem \mathbb{P} stemming from local relationships, can be expressed in terms of a functional equation $\Phi^{(\xi)} = 0$, where the field ξ is the unknown, and where Φ represents the differential relationships derived from physical considerations, that a field ξ should satisfy to be the solution of \mathbb{P} . Let us note here that the functional equation $\Phi^{(\xi)} = 0$ merely represents any differential equation, or system of equations, over a field ξ of one or more dimensions. In our example, the field ξ is the association of an electrostatic potential $V(x)$ to each point x . The equation that has to be satisfied is $\forall x, \Delta V(x) - \rho(x) = 0$. This is better expressed by the corresponding functional equation, $\Delta V - \rho = 0$, where the whole mapping V is the unknown.

Φ can thus be defined as follows, where $p \in \mathbb{N}$ can be thought of as the number of independent real equations necessary to express the local relationships which are to be satisfied at any point of \mathbb{R}^m ($p = 1$ in our example since only one scalar equation describes the problem):

$$\Phi : (\mathbb{R}^m)^{\mathbb{R}^n} \mapsto (\mathbb{R}^m)^{\mathbb{R}^p} \\ \xi \rightarrow \Phi^{(\xi)}$$

In other words, $\Phi^{(\xi)}$ is a mapping of a vector of p real values over the physical space \mathbb{R}^m . For any point $x \in \mathbb{R}^m$ in space, the i^{th} component of $\Phi^{(\xi)}(x) \in \mathbb{R}^p$, which would be zero if ξ was the solution of \mathbb{P} , actually corresponds to the local amount of violation of the i^{th} real local equation used to describe the problem at x : in our example, $\Delta V(x) - \rho(x)$, if not null, is the violation of Poisson equation at x .

Using a functional equation instead of considering ξ as a given numerical instantiation, leads us to stress that the mapping $\Phi^{(\xi)}$ depends on ξ intrinsically, whatever its actual instantiation, or value, is. The functional formalism allows to handle the dependency itself, *i.e.* the way all violations $\Phi^{(\xi)}$ over the physical space \mathbb{R}^m depend on the whole field ξ .

The original idea of this paper is that this formalism allows to express, and exploit, the variations of $\Phi^{(\xi)}$ when ξ changes, by using functional derivatives of the mapping Φ with respect to ξ , for driving the resolution process — *i.e.* for finding ξ^* for which $\Phi^{(\xi^*)} = 0$. Once again, let us note here that the functional derivatives are not as mathematically exotic as they may seem: they simply correspond to the derivative of one side of a differential equation with respect to the unknown field itself. In our example, this means deriving $\Delta V - \rho$ with respect to V .

To make this approach computationally tractable, we need to discretize the problem. This is performed by discretizing Φ on a finite mesh $\Omega \subset \mathbb{R}^m$, the discretized

problem being then expressed as $\tilde{\Phi}^{(\tilde{\xi})} = 0$, where $\tilde{\xi}$ is the unknown and $\tilde{\Phi}$ is defined as follows:

$$\tilde{\Phi} : (\Omega)^{\mathbb{R}^n} \mapsto (\Omega)^{\mathbb{R}^p} \\ \tilde{\xi} \rightarrow \tilde{\Phi}^{(\tilde{\xi})}$$

Usually, Ω will often be a set of regularly spaced points which is supposed to describe \mathbb{R}^m with enough precision, as is often done for the numerical solving of differential equations. Moreover, if a boundary condition stands at some $\omega \in \Omega$, we have $\tilde{\Phi}^{(\tilde{\xi})}(\omega) = 0, \forall \tilde{\xi}$. It means that the satisfaction of the equations at boundary conditions has to be ensured by the formulation of $\tilde{\Phi}$ itself, whatever $\tilde{\xi}$. Having boundary conditions hard-wired at the level of functional $\tilde{\Phi}$ in our formulation is of primary importance in the resolution process, as detailed further.

In our example, let us solve the 1D Poisson equation on a monodimensional mesh Ω of N regularly spaced points x_1, \dots, x_N . In the following, the value $V(x_i)$ associated by the mapping V at the point x_i will be shortened to V^i . The same stands for the charge $\rho(x_i)$ at the point x_i , that will be written as ρ^i . The discretized problem can then be found by finite difference as the following, provided V^1 and V^N are defined as boundary conditions and d is the sampling step :

$$\forall i \in \mathbb{N}, 1 < i < N, \frac{1}{d^2} (V^{i-1} - 2V^i + V^{i+1}) - \rho^i = 0.$$

Once again, let us stress here that the whole expression, including all space points is seen as depending on a *single* functional parameter V , which is a function over the discrete set $\{x_1, \dots, x_N\}$. This function V is what is actually generally formalized above as $\tilde{\xi}$.

B. General method

Getting back to the general case, solving the problem means finding ξ^* for which $\Phi^{(\xi^*)} = 0$, which means finding a field $\tilde{\xi}$ for which $\tilde{\Phi}^{(\tilde{\xi})}$ is as close to the 0 mapping as possible given a distance on the functional space $(\Omega)^{\mathbb{R}^p}$. This, in turn, is equivalent to zeroing all p relations $\tilde{\Phi}^{(\tilde{\xi})}(\omega)$ for *all* $\omega \in \Omega$. Finally, this can be equivalently done by similarly zeroing

$$\mathcal{E}^{(\tilde{\xi})} = \sum_{\omega \in \Omega} \left| \tilde{\Phi}^{(\tilde{\xi})}(\omega) \right|$$

where $|\cdot|$ is any given norm on \mathbb{R}^p . Let us note that $\mathcal{E}^{(\tilde{\xi})}$ can here be understood as a *functional* that measures the sum of the violations of the physical relationships $\tilde{\mathbb{P}}$ by a field $\tilde{\xi}$ given as the formal parameter. In other words, equation (2) means that a given discretized differential problem is solved by a given field $\tilde{\xi}$ if, and only if, at each point in space, the values of the field meet the differential problem.

The computation of $\mathcal{E}(\tilde{\xi})$ produces a scalar from a given state $\tilde{\xi}$ of the discretized problem variables. This scalar can be viewed as an *evaluation* of this state. For further purpose, let us define more generally an evaluation as a function $\zeta \in \left((\Omega)^{\mathbb{R}^n}\right)^{\mathbb{R}}$. \mathcal{E} is precisely an *evaluation* that is suited for quantifying the satisfaction of $\tilde{\Phi}$ by some state $\tilde{\xi}$.

In our example, if the norm is chosen as the simple square, equation (2) translates to

$$\mathcal{E}^{(V)} = \sum_{i=2}^{N-1} \left(\frac{1}{d^2} (V^{i-1} - 2V^i + V^{i+1}) - \rho^i \right)^2.$$

As mentioned previously, we have to set $\tilde{\Phi}$ so that it forces *intrinsically* the satisfaction of differential equations at boundary conditions. This has been done here easily by just a priori removing boundary terms 1 and N from the sum, because their values are known from the Dirichlet conditions and thus no error can be committed on them. As the resolution process described below is grounded on the formulation of this error, boundary conditions will implicitly influence the resolution process via our specific formulation of error.

The straightforward method for numerically evaluating the solution $\tilde{\xi}^*$ to $\tilde{\mathbb{P}}$, *i.e.* the value of $\tilde{\xi}$ that best zeroes —*i.e.* that *minimizes*— $\mathcal{E}(\tilde{\xi})$, is the standard Newton method applied to a multidimensional optimization problem. Let us note here however that this minimization process does not ensure the zeroing of $\mathcal{E}(\tilde{\xi}^*)$, which is to be verified *a posteriori* by evaluating $\mathcal{E}(\tilde{\xi}^*)$.

To undertake this optimization task, we previously need to define a canonical basis of the functional space $(\Omega)^{\mathbb{R}^n}$ with respect to which the *gradient* and *Hessian* will be taken. If δ is the Kronecker symbol and $\{r\}_i$ is the canonical basis of \mathbb{R}^n , let us define $\{e\}_{(\omega,i)}$, the canonical basis of $(\Omega)^{\mathbb{R}^n}$ as the set of functions $e_{(\omega,i)}$, for all $\omega \in \Omega$ and all $1 \leq (i \in \mathbb{N}) \leq n$:

$$e_{(\omega,i)} : \begin{array}{l} \Omega \mapsto \mathbb{R}^n \\ \omega' \rightarrow \delta_{\omega\omega'} r_i \end{array} \quad (3)$$

The partial derivative of an evaluation ζ at point $\tilde{\xi}$ according to basis vector $e_{(\omega,i)}$ is by definition $\lim_{h \in \mathbb{R} \rightarrow 0} \left(\zeta(\tilde{\xi} + h e_{(\omega,i)}) - \zeta(\tilde{\xi}) \right) / h$. This value is, by definition of the gradient, the actual (ω, i) component of $\text{grad}_{\{e\}_{(\omega,i)}}^{(\zeta)}(\tilde{\xi})$. Such consideration supports two major aspects in the resolution process presented in this paper. First, gradients are used to compute a rule for iteratively driving current state toward the solution of the differential problem. This is discussed next in this section. Secondly, it can be noticed that having $\forall \tilde{\xi}, \text{grad}_{\{e\}_{(\omega,i)}}^{(\zeta)}(\tilde{\xi}) = 0$ means that the i -th variable at point ω , that is the i -th component of $\tilde{\xi}(\omega)$, is not used in the computation of evaluation ζ . This helps to

identify the variables that are useful for some evaluation, which is central for ensuring the locality of our resolution process, as will be discussed further in next section.

In our example, the basis vector $e_{(\omega,i)}$ is reduced to $e_{(\omega,1)}$ since $p = 1$. As ω is a given x_i , this basis vector is the mapping with 0 potential everywhere, except at x_i where the value V^i equals 1. Let us write this $e_{(x_i,1)}$ as v_i .

Using these definitions of derivation and getting back to the general case, the Newton method consists in building a series $\tilde{\xi}_t$ defined as follows, the limit of which should be the sought solution $\tilde{\xi}^*$ to $\tilde{\mathbb{P}}$, the field which is solution to our initial differential problem:

$$\begin{aligned} \tilde{\xi}_{t+1} &= \tilde{\xi}_t - \mu_{\{e\}_{(\omega,i)}}^{(\mathcal{E})}(\tilde{\xi}_t) \\ \mu_{\{e\}_{(\omega,i)}}^{(\mathcal{E})}(\tilde{\xi}_t) &= \bar{\text{H}}_{\{e\}_{(\omega,i)}}^{(\mathcal{E})}(\tilde{\xi}_t) \cdot \text{grad}_{\{e\}_{(\omega,i)}}^{(\mathcal{E})}(\tilde{\xi}_t) \end{aligned} \quad (4)$$

where $\bar{\text{H}}$ is the inverse of the Hessian matrix.

The above expression requires some derivability conditions on \mathcal{E} , and thus on both $\tilde{\Phi}$ and the chosen norm on \mathbb{R}^n . The former is assumed, since it stems from the problem \mathbb{P} itself: the differential problem is here assumed to be derivable with respect to the unknown field. The latter is ensured by the appropriate choice of the used norm. As another precaution to be taken on that choice, the used norm must ensure that no component of the gradient — and thus of the Hessian inverse — neither supersedes the others nor is superseded by them, for this is known to create stability problems in the iteration defined by (4). The conventional $\| \cdot \|_2$ norm, or its square, is for instance a good choice, provided \mathbb{P} is conveniently normalized, *i.e.* that the unknown of the initial differential problem is a normalized quantity which has an order of magnitude around 1.

Equation (4) can be applied to our example by simply replacing $\tilde{\xi}_t$ by V_t and $\{e\}_{(\omega,i)}$ by the set of all $\{v\}_i$. This yields a complex expression for $\mu_{\{v\}_i}^{(\mathcal{E})}(V_t)$, too complicated to show here, that involves all $V^i, 1 \leq i \leq N$.

C. Local only computations

The effective computation of such a series as defined by (4) implies to compute, for each step t , the gradient and inverse Hessian with respect to $\{e\}_{(\omega,i)}$, which implies getting access to the whole Ω , in contradiction with our initial goal which was to design a *local-only* computational method. To overcome this limitation, we present in the following a method inspired from the stochastic gradient descent method [27], the locality of which will be established in the next section. The stochastic gradient method consists in updating $\tilde{\xi}$ by considering only a few of its components at a time. We choose to consider a single ω in Ω at each step, thus modifying

only $\tilde{\xi}^\omega$, the ω -related components of $\tilde{\xi}$, *i.e.* the values of the field at a given point in the mesh. Therefore, the gradient and Hessian appearing in (4) are taken not with respect to the whole $\{e\}_{(\omega,i)}$ but rather with a subset $\{e\}_\omega$ of it, restricted to ω , defined as the set $\{e_{(\omega',i)} : \omega' = \omega \text{ and } 1 \leq i \leq n\}$. The system of interdependent equations resulting from the problem discretization is thus derived with respect to the field values at a given point at a time only. One such step is therefore defined as follows:

$$\tilde{\xi}_{t+1}^\omega = \tilde{\xi}_t^\omega - \mu_{\{e\}_\omega}^{(\mathcal{E})} \left(\tilde{\xi}_t \right), \quad (5)$$

which, in the frame of our example, translates to:

$$V_{t+1}^i = V_t^i - \mu_{|v_i}^{(\mathcal{E})}(V_t).$$

The above relationship describes a series for a given point ω of Ω . For the series (4) to be completely approximated by the stochastic method, the relationship (5) is of course to be iterated over Ω with a random choice of $\omega \in \Omega$ at each step: for the derivative to be complete, it is here taken successively with respect to the field values at each point in the mesh. Thus, provided $\mu_{\{e\}_\omega}^{(\mathcal{E})}$ is somehow local, an issue that will be addressed in the next section, the above considerations allow to consider (5), at ω , as the definition of a *continuous* automaton, or CNN, which is an extension of classical cellular automata for which the cell states are allowed to take their values in \mathbb{R}^n . This automaton can be implemented for any given differential problem \mathbb{P} by evaluating $\mu_{\{e\}_\omega}^{(\mathcal{E})}$ for this particular problem. Evaluating $\mu_{\{e\}_\omega}^{(\mathcal{E})}$ can be done, as equation (4) suggests, by taking the proper gradient and Hessian of the discretized problem at each point in the mesh. Applied to our example, this method allows to calculate, for each point in the monodimensional mesh, the update rule to be applied to that point. 4 different rules are found, which are given in table I.

As can also be inferred from table I, the automaton described by (5) for each ω departs from the strict definition of a cellular automaton by the fact that the update rule for all cells ω are only the same for a vast majority of them, but not strictly all. Indeed, because of the existing boundary conditions, the H and grad operators will not give the same result for all points, since the boundary conditions are considered as constants. Hence, a Dirichlet boundary is described in the automaton by a constant cell, the value of which is given by the automaton initial state.

At this point of the paper, we have defined an automaton that can be automatically generated from any given differential problem, thanks to automated formal derivative computing that applies equation (5) at each point of the mesh. To ensure the computational efficiency of the automaton, the next section is devoted to showing that

equation (5) actually defines an automaton whose cells can be updated using the values of their neighbors only, so that an implementation on a local classical Cellular Automaton can be undertaken.

III. LOCALIZATION OF EACH CELL NEIGHBORHOOD

A. Neighborhood definition

The definition of the *neighborhood* $\mathcal{V}(\zeta)$ of a given *evaluation* ζ (see section II B for a definition) is to be understood as being the set of all the points ω needed in the computation of ζ .

$$\begin{aligned} \mathcal{V} : \left((\Omega)^{\mathbb{R}^n} \right)^{\mathbb{R}} &\mapsto \mathcal{P}(\Omega) \\ \zeta &\rightarrow \left\{ \omega \in \Omega : \exists \tilde{\xi} : \text{grad}_{\{e\}_\omega}^{(\zeta)} \left(\tilde{\xi} \right) \neq 0 \right\} \end{aligned} \quad (6)$$

The automaton described in the previous section is thus practically usable if the calculations needed to evaluate it are *local*, *i.e.* expression (5) can be evaluated without requiring access to Ω as a whole. This can be formally stated as $\mathcal{V} \left(\left| \mu_{\{e\}_\omega}^{(\mathcal{E})} \right| \right) \neq \Omega$. This can happen only if some kind of locality condition on \mathbb{P} is assumed, *i.e.* if the initial differential problem is expressed in a local manner, as it is usually the case. For instance, in the frame of our example, the values of $\mathcal{V} \left(\left| \mu_{|v_i}^{(\mathcal{E})} \right| \right)$ for all points in the mesh are given in table II. In other words, the last row of table II and the last row of table I mean that only the reading of potential at $x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}$ is required to update the potential at x_i .

B. Neighborhood size

To show that the automaton is indeed local in the general case, let us first consider the specific case of the evaluation $\left| \tilde{\Phi}(\omega) \right|$, that is the error measurement at point ω . The global error evaluation \mathcal{E} is a summation of such terms (see (2)). For further use, let us define the *dependency* $\mathcal{D}(\omega, \tilde{\Phi})$ of a given $\omega \in \Omega$ involved in a problem $\tilde{\Phi}$ as the set of point ω' for which ω belongs to the neighborhood of ω' :

$$\mathcal{D}(\omega, \tilde{\Phi}) = \left\{ \omega' : \omega \in \mathcal{V} \left(\left| \tilde{\Phi}(\omega') \right| \right) \right\}.$$

Given the definition (4) of $\mu_{\{e\}_\omega}^{(\mathcal{E})}$, the gradient can be linearly distributed over the additive components of \mathcal{E} as in (8). The summation term appearing in (7) has been restricted to those ω' in Ω for which the gradient does not vanish, *i.e.* those $\omega' \in \mathcal{D}(\omega, \tilde{\Phi})$. The summations

$i = 1$ or $i = N$	$V_{i+1}^i = V_i^i$
$i = 2$	$V_{i+1}^i = \frac{1}{5} (2V_i^{i-1} + 4V_i^{i+1} - V_i^{i+2} + d^2 * (\rho^{i+1} - 2\rho^i))$
$i = N - 1$	$V_{i+1}^i = \frac{1}{5} (2V_i^{i+1} + 4V_i^{i-1} - V_i^{i-2} + d^2 * (\rho^{i-1} - 2\rho^i))$
$3 \leq i \leq N - 2$	$V_{i+1}^i = \frac{1}{6} (-V_i^{i-2} + 4V_i^{i-1} + 4V_i^{i+1} - V_i^{i+2} + d^2 * (\rho^{i-1} - 2\rho^i + \rho^{i+1}))$

TABLE I: Update rules for the monodimensional automaton which solves the monodimensional Poisson equation, as computed from (5).

$i = 1$ or $i = N$	$\{x_i\}$
$i = 2$	$\{x_{i-1}, x_{i+1}, x_{i+2}\}$
$i = N - 1$	$\{x_{i-2}, x_{i-1}, x_{i+1}\}$
$3 \leq i \leq N - 2$	$\{x_{i-2}, x_{i-1}, x_{i+1}, x_{i+2}\}$

TABLE II: Neighborhoods $\mathcal{V} \left(\left| \mu_{\{v\}_i}^{(\mathcal{E})} \right| \right)$ for all points of a cellular automaton which solves the monodimensional Poisson Equation

product in (8) is obtained by similarly distributing the Hessian.

$$\begin{aligned}
\mu_{\{e\}_\omega}^{(\mathcal{E})} &= \sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \bar{H}_{\{e\}_\omega}^{(|\mathcal{E}|)} \text{grad}_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \quad (7) \\
&= \bar{H}_{\{e\}_\omega}^{(|\mathcal{E}|)} \sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \text{grad}_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \\
&= \left(\sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} H_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \right)^{-1} \sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \text{grad}_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \quad (8)
\end{aligned}$$

The neighborhood of a product being included in the union of its operands neighborhoods, from (6), the neighborhood of $\mu_{\{e\}_\omega}^{(\mathcal{E})}$, according to (8), can be limited to

$$\begin{aligned}
\mathcal{V} \left(\left| \mu_{\{e\}_\omega}^{(\mathcal{E})} \right| \right) &\subset \mathcal{V} \left(\left| \sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} H_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \right|^{-1} \right) \\
&\cup \mathcal{V} \left(\left| \sum_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \text{grad}_{\{e\}_\omega}^{(|\tilde{\Phi}(\omega')|)} \right| \right) \quad (9)
\end{aligned}$$

From definition (6), it can be shown that the neighborhood of a derivative, or a gradient, is included in the neighborhood of its operand. The same holds for the neighborhood of a Hessian since any line or column of the Hessian is a derivative of the gradient. Therefore, the right-hand term of the above union is included in $\bigcup_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \mathcal{V} \left(\left| \tilde{\Phi}(\omega') \right| \right)$.

Furthermore, the neighborhood of a matrix norm $|M|$ is obviously included in the union of the neighborhoods of all its components. The same holds for the inverse matrix $|(M)^{-1}|$ since each of its components can be obtained by a combination of the components of M . We can therefore conclude that the left-hand term of the union in (9) is also a subset of $\bigcup_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \mathcal{V} \left(\left| \tilde{\Phi}(\omega') \right| \right)$.

Therefore, provided we can assume that $\mathcal{V} \left(\left| \tilde{\Phi}(\omega) \right| \right)$ is small enough for all $\omega \in \Omega$ —which is ensured if the differential problem \mathbb{P} is defined locally—, the calculations to be undertaken to evaluate $\mu_{\{e\}_\omega}^{(\mathcal{E})}$ for each cell ω are local to some extended neighborhood of that cell:

$$\mathcal{V} \left(\left| \mu_{\{e\}_\omega}^{(\mathcal{E})} \right| \right) \subset \bigcup_{\omega' \in \mathcal{D}(\omega, \tilde{\Phi})} \mathcal{V} \left(\left| \tilde{\Phi}(\omega') \right| \right)$$

From the definition of the neighborhood, the above inclusion means that the calculations involved in computing the update rule $\mu_{\{e\}_\omega}^{(\mathcal{E})}$ at a given point in the mesh only involve the field values of the dependent points ω' in the sense of \mathcal{D} , the actual number and repartition of those points being dependent on the differential problem itself.

Therefore, any differential problem defined locally can by this method be transformed into a cellular automaton involving local computations only, the fixed point of which is the quantitative solution to the differential problem. This is of primary importance if the computation is to be parallelized on clusters, for which having local computation is the key to efficiency. This latter point, currently at work, is out of the scope of this paper.

In the frame of our example, the automaton obtained by our formal resolution process is given in table I.

IV. APPLICATION EXAMPLES

As hinted before, we have implemented the continuous automaton described in the previous sections with the help of an off-the-shelf formal computing software, essentially used to formally evaluate the update rule from the differential problem through equation (5), and a cellular automata environment analogous to those reported in [25, 26]. We have thus automated the computation

from the specification of the discrete differential problem $\tilde{\mathbb{P}}$ to the design of the adequate continuous automaton, the fixed point of which is the solution to $\tilde{\mathbb{P}}$.

Let us now illustrate this process with two examples. The first one is the generalization to 3 dimensions of the example used in the previous sections. The monodimensional example was trivial, as it possessed a straightforward solution. The 3D one is a little trickier but can still be solved numerically by other known methods. That is the reason why a more physically realistic example will be shown: the application of strictly the same method to the non-paraxial beam propagation equation, which is not so easy to solve numerically.

A. Poisson equation

The first example, simple and academic, is thus the solving of Poisson Equation for V in the three dimensions of space: $\Delta V(x, y, z) = \rho(x, y, z)$ for any given ρ , the Dirichlet boundary conditions being set on the sides of the computing cube window. The corresponding discrete problem is straightforward and is obtained through finite difference centered second derivatives on each dimensions of space, for the same space step d .

The automaton obtained through the evaluation of (5) on each point of the mesh has 28 different update rules. The update rule obtained for ω such as the boundaries conditions are not in $\mathcal{V} \left(\mu_{\{e\}, \omega}^{(\mathcal{E})} \right)$ concerns the vast majority of the mesh nodes ω and is shown below. It is a centro-symmetric three dimensional convolution kernel involving V and ρ . Only middle and lower parts of these kernels are shown, the upper part being obtained by symmetry.

$$V \leftarrow \frac{1}{42} \left[V \left(\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right) + d^2 \cdot \rho \left(\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right) \right].$$

The 27 other update rules account for the boundary conditions. When launched, the system converges to a fixed point, corresponding to the result that, in that case, can also be obtained with more conventional methods.

As stated above, the result has to be checked valid *a posteriori* by evaluating the remaining error as defined by (2). For a better assessment of the performance, we will provide two values of the remaining error. The first one is the *mean error*, which is simply $\mathcal{E}(\tilde{\xi})$ when $\tilde{\xi}$ takes the value of the solution found, all divided by the number of points, to get the mean error per mesh point. The other one, the *maximum error*, is defined as $\mathcal{E}_M(\tilde{\xi}) = \max_{\omega \in \Omega} \left| \tilde{\Phi}(\tilde{\xi})(\omega) \right|$ and yields the maximum er-

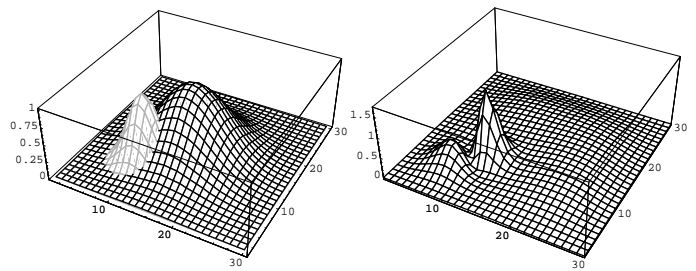


FIG. 1: Left : laser beam Gaussian profile to be coupled in the waveguide (black) and waveguide (grey) (A.U.). Right : beam profile after a 3mm propagation. The window size is $30\mu\text{m}$ and the beam wavelength is $250\mu\text{m}$. The highest peak on the right evidences the light which is coupled into the waveguide.

ror per point. Both will be normalized to the maximum component of $\tilde{\xi}$.

For a $20 \times 20 \times 20$ mesh and for a value of ρ varying from 1 to 0 from one side of the cube to the other, the *a posteriori* computed mean and maximum errors are 7×10^{-3} and 4×10^{-1} respectively for $d = 1$ and decrease with it. The maximum error, that can seem large, is due to strong gradients in the solution close to the boundary conditions and the very crude mesh used. The strong gradients are caused by the non realistic values taken for ρ . However, the mean error shows that the solution found, aside from a few points, is still acceptable, despite the sparse mesh.

B. Non paraxial laser beam propagation

Furthermore, as we will show now, a better assessment of the performance of this method will be established by solving a physically realistic problem, which is not so easy to solve by other conventional methods. Indeed, we now aim to compute the coupling of a Gaussian laser beam of width W into a Gaussian shaped waveguide of width $\frac{W}{2}$ and modulation depth 10^{-4} . The centers of both beam and waveguide are set to a distance of $\frac{W}{2}$, both being aligned in the same direction. In the computation process, we will *not* make the standard simplifying paraxial approximation, which makes our problem difficult to solve by conventional methods.

The non-paraxial propagation equation to be solved is thus the following, where A is the wave electric field to be found, z is the propagation direction, k is the wave vector, n and δn are the given refraction index and a small variation of it:

$$\frac{\partial A}{\partial z} - \frac{i}{2k} \Delta A = \frac{ik}{n} \delta n A. \quad (10)$$

The problem is solved by deriving two real equations from (10), discretizing them with finite difference centered derivatives except along z where left-handed deriva-

tives are needed because of the impossibility to give a boundary condition on one side of the propagation axis.

The adequate continuous automaton (it also has 28 update rules but is too complicated to show here) is then computed from the discretized problem. When launched on a $30 \times 30 \times 30$ network, it stabilizes to a fixed point shown on figure 1, where the light is found to be coupled into the waveguide. The *a posteriori* remaining mean and maximum errors are now computed to be 2×10^{-12} and 9×10^{-12} respectively, proving that the obtained solution does indeed meet the differential problem requirements.

V. CONCLUSION

We have described and successfully assessed what we believe to be an original method allowing to tailor a Cellular Neural Network so that its fixed point *quantitatively* solves a given differential problem. We believe that this

method can be applied to most continuous differential problems. Since time and space are considered equally, it does not have the stability issues encountered in classical CNNs [8], as the dynamic process described in this paper is an explicit *energy* minimization, the energy considered here being the error \mathcal{E} .

An accompanying formal computing automation can thus offer the unique possibility to reduce differential problem solving to the mere specification of the problem with an adequate formal language and its feeding to a specifically designed Continuous Automaton. This work can therefore be used for the design of an automated piece of software able to solve differential problems while sparing the user the need to get involved in actual numerical mathematics and computer programming, thus sparing code development time. This can be particularly useful for the simulation of new differential problems for which no off-the-shelf software is available.

-
- [1] A. W. Burks, *Von Neumann's Self-reproducing Automata* (University of Michigan, 1969).
 - [2] S. Wolfram, *Rev. Mod. Phys.* **55**, 61 (1983).
 - [3] S. Wolfram, *A new kind of science* (Wolfram Media, Champaign, 2002).
 - [4] J. R. Weimar and J. P. Boon, *Phys.Rev.E* **49**, 1749 (1994).
 - [5] B. Drossel and F. Schwabl, *Phys. Rev. Lett.* **69**, 1629 (1992).
 - [6] D. H. Rothman and S. Zaleski, *Rev. Mod. Phys.* **66**, 1417 (1994).
 - [7] L. O. Chua and L. Yang, *IEEE Transactions on Circuits and Systems* **35**, 1257 (1988).
 - [8] T. Roska, L. O. Chua, D. Wolf, T. Kozek, R. Tetzlaff, and F. Puffer, *IEEE Transactions on Circuits and Systems-I: Fundamental theory and applications* **42**, 807 (1995).
 - [9] A. Lonkar, R. Kuntz, and R. Tetzlaff, in *6th EEE International Workshop on Cellular Neural Networks and Their Applications* (2000), pp. 123–128.
 - [10] F. Sargeni and V. Bonaiuto, *Analog Integrated Circuits and Signal Processing* **44**, 283 (2005).
 - [11] A. Slavova, *Journal of Computational and Applied Mathematics* pp. 387–404 (2000).
 - [12] K. Hadad and A. Piroozmand, *Annals of Nuclear Energy* (2007).
 - [13] O. Bandmann, *Future Generation Computer Systems* pp. 737–745 (2002).
 - [14] T. Kozek, L. O. Chua, T. Roska, D. Wolf, R. Tetzlaff, F. Puffer, and K. Lotz, *IEEE Transactions on Circuits and Systems-I: Fundamental theory and applications* **42**, 807 (1995).
 - [15] A. Slavova and P. Zecca, *Journal of Computational and Applied Mathematics* pp. 13–24 (2003).
 - [16] N. R. S. Simons, G. E. Bridges, and M. Cuhaci, *J. Comput. Phys.* **151**, 816 (1999), ISSN 0021-9991.
 - [17] X. Zhou, B. Liu, and B. Shi, in *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics* (2003), vol. V of *Computer Science and Engineering: I*, pp. 240–244.
 - [18] V. N. Vapnik, *The Nature of Statistical Learning Theory*, *Statistics for Engineering and Information Science* (Springer, 2000).
 - [19] T. Tokihiro, D. Takahashi, J. Matsukidaira, and J. Satsuma, *Phys Rev. Lett.* **76**, 3247 (1996).
 - [20] A. Doeschl, M. Davison, H. Rasmussen, and G. Reid, *Math. Comp. Mod.* **40**, 977 (2004).
 - [21] W. Kunishima, A. Nishiyama, H. Tanaka, and T. Tokihiro, *Journ. Phys Soc. Japan* **73**, 2033 (2004).
 - [22] S. Omohundro, *Physica D* **10D**, 128 (1984).
 - [23] G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing Partial Differential Equations and the Calculus of Variations*, vol. 147 of *Applied Mathematical Sciences* (Springer, 2006), 2nd ed.
 - [24] C. Rekeczky, *International Journal of CVircuit Theory and Applications* pp. 313–348 (2002).
 - [25] M. Cannataro, S. D. Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia, *Parallel Computing* **21**, 803 (1995).
 - [26] G. Spezzano and D. Talia, in *Virtual shared memory for distributed architectures* (Nova Science Publishers, Inc., Commack, NY, USA, 2001), pp. 51–68.
 - [27] J. C. Spall, *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control* (Wiley-Interscience, 2003).