2D Simplified Wildfire Spreading Model in Python: From *NumPy* to *CuPy*

Daniel San Martin
Universidad Técnica Federico Santa María, Departamento de Informática,
Valparaíso, Chile, 2390123
daniel.sanmartinr@usm.cl

and

Claudio E. Torres
Universidad Técnica Federico Santa María, Departamento de Informática,
Valparaíso, Chile, 2390123
ctorres@inf.utfsm.cl

Abstract

Wildfires are a problem of great interest in the world because of the damage they cause every year to forest, wild fauna and flora, an also threatening human lives, among others. There exist some computational models for numerical simulations to address this phenomena, and a few of them have an open-source implementation. The goal of this work is to extend a *CPU Python*'s implementation of wildfire open-source framework developed in *NumPy*, to a *GPU* improved version using *CuPy*. The algorithm used is based on a numerical discretization of a wildfire spreading mathematical model described by a system of partial differential equations. Computational and mathematical components, numerical simulations and applications are described in details in the document. In addition, this work includes a brief performance comparison between both implementations, pointing out that we can achieve lower execution times using the *CuPy GPU* implementation without spending enough programming effort.

Keywords: Wildfire Modeling, Scientific Computing, Numerical Methods, *NumPy*, *CuPy*, *Python*, *GPU*.

1 Introduction

Wildfires are ubiquitous phenomena of global interest since they may occur in almost every zone of the world [1], and have the potential to damage large extensions of land [2]. Fire can start for different causes, including natural causes such as lightning, spontaneous combustion, and volcanic eruption, but about 75% of worldwide fires are caused by humans, either deliberately or by negligence [3, 4]. Due to the devastating effects of this type of disaster and their dynamics behavior, it is crucial to have open-source implementations as an active tools for analysis of wildfires, specifically, based on fire models.

According to [5] fire models can be grouped into three categories: risk assessment, propagation and effect models. Risk assessment is associated to quantification of possible fire episodes. The second category study the propagation of fire, starting from physical laws up to regression models. Finally, effect models is a high level analysis, including fuel management planning, tree mortality up to analysis of ecosystems, among others. These categories are not mutually exclusive. Propagation models can be sub-classified into three types: empirical, physical and semi-empirical. The empirical models use a statistical approach to describe the fire behavior. Physical models are based exclusively in physical laws. And finally, semi-empirical models combine both empirical and physical approaches.

The goal of this work is extend a *CPU* implementation developed in *Python* using *Numpy* and described in [6], to a *GPU* implementation prototype exploring the use of *CuPy*.

2 Related work

During the last decade, two of the most used approaches in Wildfire Modeling are *Partial Differential Equations (PDE)* and *Cellular Automata (CA)* based models. The first class of models is based on physical laws, describing physical and chemical processes using differential equations [7]. In the second kind of models, the fire dynamic is studied on a landscape grid following transition rules. These rules determine the state of each cell and they are based on theoretical and semi-empirical mathematical fire behavior models [7]. Initial efforts of *PDE*-based modeling for fire spreading were the work of Albini [8] and Weber [9]. In the area of coupled atmosphere-fire models are the work of Grishin [10], Linn with *FIRETEC* [11, 12], and McGrattan with *FDS* applied to wildland fires [13, 14], among others. Regarding empirical models we can mention *FARSITE* [15] or *ELMFire* [16], but as well as coupled models, they are outside the scope of this article. For more details of forest fire models see [7] or [17].

This article presents a *PDE*-based spreading model which uses a similar mathematical formulation as presented in [18], [19], and [20]. Asensio & Ferragut [18] presented a 2-dimensional model which describes the dynamic of temperature and solid fuel. They solved the equations using a Finite Element Method (*FEM*) for the spatial discretization and a semi-implicit Euler scheme for time integration. This work presented some numerical results qualitatively consistent with fires under controlled conditions. Mandel et al. [19] derived a similar model than Asensio & Ferragut, based on the conservation of energy equations and the balance of fuel supply. They used Finite Difference Method (*FDM*) for the spatial derivatives and Explicit Euler Method for time integration. As an active fire simulation tool, this work introduces data assimilation techniques to correct fire dynamics using real measurements. Finally, Eberle et al. [20] also derived a similar model to the previous ones, using the conservation of mass, energy and fuel supply balance equation. The equations were solved using a Radial Basis Function (*RBF*) Collocation Method for the spatial discretization, and a Crank-Nicolson scheme for time integration. This work shows the ability of the model to simulate fire for 2 different fuel types and how the propagation speed varies according to the parameters used.

A selection of related work using *CA* and *PDE* approaches and their code availability is presented in Table 1.

Table 1: Summary of models related to this work.

Model Approach	Work	Code Reproducible
CA	Hansen (1993) [21]	✓
	Karafyllidis et al. (1997) [22]	×
	Quartieri et al. (2010) [23]	×
	Almeida et al. (2011) [24]	×
	Ghisu et al. (2015) [25]	×
	Fernandez-Anez et al. (2017) [26]	×
PDE	Montenegro et al. (1997) [27]	×
	Asensio et al. (2002) [18]	×
	Ferragut et al. (2006) [28]	×
	Ferragut et al (2007) [29]	×
	Mandel et al. (2008) [19]	×
	Ferragut et al. (2015) [30]	×
	Eberle et al. (2015) [20]	×
	This work	✓

3 Methods

3.1 Wildfire mathematical model

The mathematical model used for this work is based on the proposed by [18] with the simplification introduced by [20].

Assuming a spatial domain $\mathbf{x} = (x, y) \in \Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{R}^2$ with boundary $\Gamma = \partial\Omega$, and a temporal domain $t \in [0, t_{\max}] \subset \mathbb{R}_0^+$. Let $u(\mathbf{x}, t)$ be the numerical value of the temperature and $\beta(\mathbf{x}, t)$ the amount of fuel, both at spatial coordinate $\mathbf{x}$ and at time t. The vector field that models the effect of wind $\mathbf{w}(t)$ and topography $Z(\mathbf{x})$ is $\mathbf{v}(\mathbf{x}, t) = \mathbf{w}(t) + \nabla Z(\mathbf{x})$, and $f(u, \beta)$ is a nonlinear heat source that describes the interaction between fuel and temperature. The simplified mathematical model of wildfire spreading, is

defined as

$$\begin{array}{rcll} u_t & = & \kappa\,\Delta u - \mathbf{v}\cdot\nabla u + f(u,\beta), & \text{in } \Omega\times(0,t_{\max}],\\ \beta_t & = & g(u,\beta), & \text{in } \Omega\times(0,t_{\max}],\\ u(\mathbf{x},t) & = & u_\Gamma(\mathbf{x},t), & \text{on } \Gamma\times(0,t_{\max}],\\ \beta(\mathbf{x},t) & = & \beta_\Gamma(\mathbf{x},t), & \text{on } \Gamma\times(0,t_{\max}],\\ u(\mathbf{x},0) & = & u_0(\mathbf{x}), & \text{in } \Omega,\\ \beta(\mathbf{x},0) & = & \beta_0(\mathbf{x}), & \text{in } \Omega, \end{array} \tag{1}$$

where,

$$f(u,\beta) = H_{pc}(u)\,\beta\,\exp\left(\frac{u}{1+\varepsilon\,u}\right) - \alpha u,$$

$$g(u,\beta) = -H_{pc}(u)\,\frac{\varepsilon}{q}\,\beta\,\exp\left(\frac{u}{1+\varepsilon\,u}\right),$$

$$H_{pc}(u) = \begin{cases} 1, & \text{if } u \geq u_{pc},\\ 0, & \text{otherwise.} \end{cases}$$

Since the model is defined in a 2-dimensional domain the gradient is defined as $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$ and the Laplace operator $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. The nondimensional parameters of the model are diffusion coefficient κ, the inverse of activation energy of fuel ε, natural convection α, reaction heat q, and the phase change threshold u_{pc}.

An assumption used in this work is that the spatial domain is large enough to avoid fire spreading at the boundary of Ω, then the following Dirichlet boundary conditions are used: $u_\Gamma(\mathbf{x},t) = \beta_\Gamma(\mathbf{x},t) = 0$ on $\Gamma\times(0,t_{\max}]$. Nevertheless, the model can be used with Neumann boundary conditions. The components used by the model are presented in Figure 1.

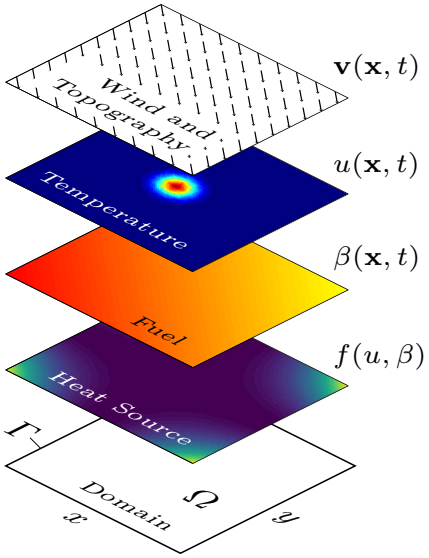


Figure 1: Sketch of model components. Figure taken from [6].

3.2 Numerical Algorithm

Numerical implementation of (1) requires the approximation of spatial and time partial derivatives involved in the problem formulation. To address these approximations, this work uses the *Method of Lines*, technique for numerical solving of *PDEs* in which all but one dimension is discretized. The following subsections will describe the methods used to approximate the partial derivatives indicated previously.

3.2.1 Method of Lines

Method of Lines (MOL) is an important approach for the numerical solving of *PDEs*. This method consists of two parts, space discretization and time integration [31]. The space discretization approximates the *PDE* by a system of *Ordinary Differential Equations (ODEs)* by discretizing the space domain using finite difference,

finite elements, spectral methods, among others. Here, time is the independent variable of the *ODE* system. In the second step, the system is solved using some numerical method for time integration, discretizing the time domain to get a fully discrete scheme.

There are several works, such as [32, 33, 31, 34], discussing the stability and convergence of the method for some particular problems. This work uses *MOL* as a mathematical framework to solve (1) numerically.

3.2.2 Spatial Approximation

In the mathematical model used for this work, temperature dynamics needs the computation of Δu and ∇u. To deal with these space partial derivatives, a discretization on a rectangular grid of N_x+1 by N_y+1 nodes is established. Let x_i and y_j the discrete version of the spatial domain, defined as

$$\begin{aligned} x_i &= x_{\min} + i\,\Delta x, \quad i = 0, 1, \ldots, N_x, \\ y_j &= y_{\min} + j\,\Delta y, \quad j = 0, 1, \ldots, N_y, \end{aligned} \tag{2}$$

where $\Delta x = (x_{\max} - x_{\min})/N_x$ and $\Delta y = (y_{\max} - y_{\min})/N_y$. Now, temperature component is defined as $u_{i,j}(t) = u(x_i, y_j, t)$ and stored in the matrix $U(t)$ described by

$$U(t) = \begin{pmatrix} u_{0,0}(t) & \ldots & u_{N_x,0}(t) \\ \vdots & \ddots & \vdots \\ u_{0,N_y}(t) & \ldots & u_{N_x,N_y}(t) \end{pmatrix}. \tag{3}$$

Following this ordering, fuel component $\beta_{i,j}(t) = \beta(x_i, y_j, t)$ is

$$B(t) = \begin{pmatrix} \beta_{0,0}(t) & \ldots & \beta_{N_x,0}(t) \\ \vdots & \ddots & \vdots \\ \beta_{0,N_y}(t) & \ldots & \beta_{N_x,N_y}(t) \end{pmatrix}, \tag{4}$$

and the vector field $\mathbf{v}(x_i, y_j, t) = (v_1(x_i, y_j, t), v_2(x_i, y_j, t)) = (v_{1\,i,j}(t), v_{2\,i,j}(t))$ is stored on the grid by the matrices $V_1(t), V_2(t)$ defined as follows

$$V_1(t) = \begin{pmatrix} v_{1\,0,0}(t) & \ldots & v_{1\,N_x,0}(t) \\ \vdots & \ddots & \vdots \\ v_{1\,N_y,0}(t) & \ldots & v_{1\,N_x,N_y}(t) \end{pmatrix}, \quad V_2(t) = \begin{pmatrix} v_{2\,0,0}(t) & \ldots & v_{2\,N_x,0}(t) \\ \vdots & \ddots & \vdots \\ v_{2\,0,N_y}(t) & \ldots & v_{2\,N_x,N_y}(t) \end{pmatrix}. \tag{5}$$

Analogous for functions $f(x_i, y_j, t) = f_{i,j}(t)$ and $g(x_i, y_j, t) = g_{i,j}(t)$,

$$F(t) = \begin{pmatrix} f_{0,0}(t) & \ldots & f_{N_x,0}(t) \\ \vdots & \ddots & \vdots \\ f_{0,N_y}(t) & \ldots & f_{N_x,N_y}(t) \end{pmatrix}, \quad G(t) = \begin{pmatrix} g_{0,0}(t) & \ldots & g_{N_x,0}(t) \\ \vdots & \ddots & \vdots \\ g_{0,N_y}(t) & \ldots & g_{N_x,N_y}(t) \end{pmatrix}. \tag{6}$$

Now, partial derivatives will be computed using the *Finite Differences Method* by means of computing the product between a *Differentiation Matrices* and the corresponding discretized variable.

Following the approach of [35], a *Central Finite Difference Method (CD FDM)* approximation for the first and second spatial partial derivative [36] is used. An extension of *Forward* and *Backward Difference* is used to keep the order of accuracy at the boundary of the domain. For more details, we included the subsection *Finite Difference formulas* in the Appendix to describe the expressions used.

For simplicity, time dependency was omitted in $u_{i,j}(t)$ and will be omitted for $U(t)$ as well. Taking into account the storage order of matrices, the boundary conditions described before, and dropping the error terms $O(\Delta x^2)$ and $O(\Delta y^2)$ spatial partial derivatives can be approximated as presented below.

- The first partial derivative with respect to x using (40), (41), and (42) is:

$$\frac{\partial u}{\partial x} \approx U D_{N_x}, \tag{7}$$

with,

$$D_{N_x} = \frac{1}{2\Delta x} \begin{pmatrix} -3 & -1 & 0 & \cdots & 0 & 0 \\ 4 & 0 & \ddots & \ddots & \vdots & \vdots \\ -1 & 1 & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & -1 & 1 \\ \vdots & \vdots & \ddots & \ddots & 0 & -4 \\ 0 & 0 & \ldots & 0 & 1 & 3 \end{pmatrix}, \tag{8}$$

- The second partial derivative with respect to x using (43), (44), and (45) is:

$$\frac{\partial^2 u}{\partial x^2} \approx U\, D^{(2)}_{N_x}, \tag{9}$$

with,

$$D^{(2)}_{N_x} = \frac{1}{\Delta x^2}\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ -5 & -2 & \ddots & \ddots & \ddots & \vdots & \vdots \\ 4 & 1 & \ddots & \ddots & \ddots & 0 & 0 \\ -1 & 0 & \ddots & \ddots & \ddots & 0 & -1 \\ 0 & 0 & \ddots & \ddots & \ddots & 1 & 4 \\ \vdots & \vdots & \ddots & \ddots & \ddots & -2 & -5 \\ 0 & 0 & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix}. \tag{10}$$

- The first partial derivative with respect to y using (46), (47), and (48) is:

$$\frac{\partial u}{\partial y} \approx D_{N_y} U, \tag{11}$$

with,

$$D_{N_y} = \frac{1}{2\Delta y}\begin{pmatrix} -3 & 4 & -1 & 0 & \dots & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 0 & 1 \\ 0 & \dots & 0 & 1 & -4 & 3 \end{pmatrix}. \tag{12}$$

- The second partial derivative with respect to y using (49), (50), and (51) is:

$$\frac{\partial^2 u}{\partial y^2} \approx D^{(2)}_{N_y} U, \tag{13}$$

with,

$$D^{(2)}_{N_y} = \frac{1}{\Delta y^2}\begin{pmatrix} 2 & -5 & 4 & -1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \dots & 0 & -1 & 4 & -5 & 2 \end{pmatrix}. \tag{14}$$

Defined these expressions, we can approximate Laplacian and gradient of u as follows,

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx U\, D^{(2)}_{N_x} + D^{(2)}_{N_y}\, U, \tag{15}$$

and,

$$\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) \approx \left(U\, D_{N_x}, D_{N_y}\, U\right). \tag{16}$$

Using (15) and (16), the system of *PDEs* defined in (1) becomes to the following dynamical system

$$\begin{aligned} \frac{dU(t)}{dt} &= \kappa\left(U(t)\, D^{(2)}_{N_x} + D^{(2)}_{N_y}\, U(t)\right) - \left(V_1(t) \odot \left(U(t)\, D_{N_x}\right) + V_2(t) \odot \left(D_{N_y}\, U(t)\right)\right) + F(t), \\ \frac{dB(t)}{dt} &= G(t), \end{aligned} \tag{17}$$

where $F(t) = f(U(t), B(t))$, $G(t) = g(U(t), B(t))$ and $\odot$ denotes the *Hadamard product* or element-wise multiplication. Notice that the values of U on Γ ensures that the boundary conditions are satisfied for each t. In particular, the Dirichlet boundary conditions $u(\mathbf{x}, t) = \beta(\mathbf{x}, t) = 0$, $\mathbf{x} \in \Gamma$, are imposed setting the values of $u_{i,j}(t) = \beta_{i,j}(t) = 0$ when $i = 0, N_x, j = 0, \dots, N_y$ and when $i = 0, \dots, N_x, j = 0, N_y$.

The next step is to solve (17) by some numerical integration method.

3.2.3 Time Approximation

This section aims to represent (17) as an *Initial Value Problem (IVP)* and solve it numerically. Rewriting (17) as

$$\begin{pmatrix}\dot{U}(t)\\ \dot{B}(t)\end{pmatrix} = \begin{pmatrix}\kappa\left(U(t)\,D_{N_x}^{(2)} + D_{N_y}^{(2)}\,U(t)\right) - \left(V_1(t)\odot\left(U(t)\,D_{N_x}\right) + V_2(t)\odot\left(D_{N_y}\,U(t)\right)\right) + F(t)\\ G(U(t),B(t))\end{pmatrix}, \tag{18}$$

with $\dot{U}(t) = \frac{dU(t)}{dt}$ and $\dot{B}(t) = \frac{dB(t)}{dt}$, a vector version of (18) is presented due to the existence of several libraries that solve *IVPs* but require a vector representation of the problem (see details in the *Vectorization* subsection from the Appendix). Let us define $\mathbf{y}(t)$ as,

$$\mathbf{y}(t) = \left(\mathrm{vec}(U(t)), \mathrm{vec}(B(t))\right)^\top, \tag{19}$$

now, (18) is redefined by the following *IVP*

$$\dot{\mathbf{y}}(t) = \Phi(t, \mathbf{y}(t)), \tag{20}$$

where,

$$\dot{\mathbf{y}}(t) = \left(\mathrm{vec}(\dot{U}(t)), \mathrm{vec}(\dot{B}(t))\right)^\top, \tag{21}$$

$$\Phi(t,\mathbf{y}(t)) = \begin{pmatrix}\mathrm{vec}\left(\kappa\left(U(t)\,D_{N_x}^{(2)} + D_{N_y}^{(2)}\,U(t)\right) - \left(V_1(t)\odot\left(U(t)\,D_{N_x}\right) + V_2(t)\odot\left(D_{N_y}\,U(t)\right)\right) + F(t)\right)\\ \mathrm{vec}\left(G(t)\right)\end{pmatrix}, \tag{22}$$

and $\mathbf{y}(0) = \left(\mathrm{vec}\left(U(0)\right), \mathrm{vec}\left(B(0)\right)\right)^\top$.

There is a wide range of numerical methods to solve (20) and it is necessary to specify a discrete version of time. This domain will be defined as,

$$t_n = t_{\min} + n\,\Delta t, \quad n = 0, 1, \ldots, N_t, \tag{23}$$

where $\Delta t = (t_{\max} - t_{\min})/N_t$. The numerical methods implemented by this work will be detailed below.

The first method used to solve (20) is *Euler Method*, mainly for first experiments and testing the convergence of spatial approximation. Defining $\mathbf{y}_n = \mathbf{y}(t_n)$, the approximation of $\mathbf{y}(t)$ at time t_n, the computation of the *IVP* using this method is

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t\,\Phi\left(t_n, \mathbf{y}_n\right). \tag{24}$$

This method has a temporal truncation error of $O(\Delta t)$ [36].

Furthermore, this work also uses *Fourth Order Runge-Kutta Method (RK4)* to solve (20), since its stability region give the flexibility of a Δt choice [35]. Defining $\mathbf{y}_n = \mathbf{y}(t_n)$, the approximation of $\mathbf{y}(t)$ at time t_n, the computation of the *IVP* using this method is

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \tag{25}$$

where,

$$\begin{aligned}
\mathbf{k}_1 &= \Phi\left(t_n, \mathbf{y}_n\right),\\
\mathbf{k}_2 &= \Phi\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2}\mathbf{k}_1\right),\\
\mathbf{k}_3 &= \Phi\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2}\mathbf{k}_2\right),\\
\mathbf{k}_4 &= \Phi\left(t_n + \Delta t, \mathbf{y}_n + \Delta t\ \mathbf{k}_3\right).
\end{aligned} \tag{26}$$

This method has a temporal truncation error of $O(\Delta t^4)$ [36].

3.2.4 Algorithm Description

Using the approach of *MOL* described in section 3.2.1, the algorithm proposed integrates spatial and time approximation presented in sub-subsections 3.2.2 and 3.2.3. The application of *RK4* with expression (25) and (22) is detailed in Algorithm 1. The approximation of Φ using expression (22) is presented in Algorithm 2.

Finally, Algorithm 3 summarize the application of *MOL* to approximate the simplified wildfire mathematical model described in subsection 3.1.

Algorithm 1 4th Order Runge-Kutta.

procedure RK4($t_n, \mathbf{y}_0, \Phi$)
  for $n = 0$ to $N_t - 1$ **do**
    $\mathbf{k}_1 \leftarrow \Phi\left(t_n, \mathbf{y}_n\right)$
    $\mathbf{k}_2 \leftarrow \Phi\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2}\mathbf{k}_1\right)$
    $\mathbf{k}_3 \leftarrow \Phi\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2}\mathbf{k}_2\right)$
    $\mathbf{k}_4 \leftarrow \Phi\left(t_n + \Delta t, \mathbf{y}_n + \Delta t\ \mathbf{k}_3\right)$
    $\mathbf{y}_{n+1} \leftarrow \mathbf{y}_n + \frac{\Delta t}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right)$
  end for
  return $\mathbf{y}_n$
end procedure

Algorithm 2 Right hand side Φ approximation.

Require: $u_\Gamma, \beta_\Gamma, \mathbf{v}, \kappa, \varepsilon, \alpha, q, u_{pc}, f, g, x_i, y_j, D_{N_x}, D_{N_y}, D^{(2)}_{N_x}, D^{(2)}_{N_y}$
procedure $\Phi(t, \mathbf{y})$
  $\mathbf{u}, \boldsymbol{\beta} \leftarrow \mathbf{y}$
  $V_1, V_2 \leftarrow \mathbf{v}(x_i, y_j, t)$
  $U \leftarrow \text{vec}^{-1}\left(\mathbf{u}\right)$
  $B \leftarrow \text{vec}^{-1}\left(\boldsymbol{\beta}\right)$
  $\dot{U} \leftarrow \kappa\left(U\, D^{(2)}_{N_x} + D^{(2)}_{N_y}\, U\right) - \left(V_1 \odot \left(U\, D_{N_x}\right) + V_2 \odot \left(D_{N_y}\, U\right)\right) + f(U, B)$
  $\dot{B} \leftarrow g(U, B)$
  Set BC with $u_\Gamma(x_i, y_j)$ and $\beta_\Gamma(x_i, y_j)$.
  $\mathbf{y} \leftarrow \left(\text{vec}\left(\dot{U}\right), \text{vec}\left(\dot{B}\right)\right)^T$
  return $\mathbf{y}$
end procedure

Algorithm 3 Main algorithm.

Require: $u_0, \beta_0, u_\Gamma, \beta_\Gamma, \mathbf{v}, \kappa, \varepsilon, \alpha, q, u_{pc}, x_{\min}, x_{\max}, y_{\min}, y_{\max}, t_{\min}, t_{\max}, N_x, N_y, N_t$
$\Delta x \leftarrow (x_{\max} - x_{\min})/N_x$
$\Delta y \leftarrow (y_{\max} - y_{\min})/N_y$
$\Delta t \leftarrow (t_{\max} - t_{\min})/N_t$
$x_i \leftarrow x_{\min} + i\,\Delta x, \quad i = 0, ..., N_x$
$y_j \leftarrow y_{\min} + j\,\Delta y, \quad j = 0, ..., N_y$
$t_n \leftarrow t_{\min} + n\,\Delta t, \quad n = 0, ..., N_t$
$U^0 \leftarrow u_0(x_i, y_j)$
$B^0 \leftarrow \beta_0(x_i, y_j)$
$\mathbf{y}_0 \leftarrow \left(\text{vec}\left(U^0\right), \left(B^0\right)\right)^T$
$\mathbf{y}_n \leftarrow \text{RK4}(t_n, \mathbf{y}_0, \Phi)$
$U^n, B^n \leftarrow$ reshape $\mathbf{y}_n$

The `reshape` procedure in Algorithm 3 is used to reconstruct the matrices according to the number of nodes N_x, N_y, and N_t. Essentially, *MOL* allows solving the *PDE* model transforming the problem into an *IVP*. This approach gives the flexibility to choose any numerical scheme for time integration and another numerical method for space derivatives. In this particular case, the proposal uses *RK4* for time and *CD FDM* for space.

The computational complexity of the entire Algorithm is $\sim 4\ N_t\,(8\,N^3 + 6\,N^2 + C_f\,N^2 + C_g\,N^2)$, where $N = N_x = N_y$ is the number of nodes in x and y direction respectively, N_t is the number of nodes in t, C_f and C_g are the costs of evaluating the function f and g. See [6] to get a detailed description of the complexity derivation.

3.2.5 Stability Analysis

To ensure the operation of the algorithm, a stability analysis to solve the problem $\dot{\mathbf{y}}(t) = \Phi(t, \mathbf{y}(t))$ is performed. Expanding the expression 18,

$$\Phi(t, \mathbf{y}) = \left(\dot{U}_{0,0}, \ldots, \dot{U}_{i,j}, \ldots, \dot{U}_{N_x,N_y}, \ldots, \dot{B}_{0,0}, \ldots, \dot{B}_{i,j}, \ldots, \dot{B}_{N_x,N_y}\right)^\top, \tag{27}$$

where,

$$\dot{U}_{i,j} = \kappa \left(\frac{U_{i+1,j} - 2\,U_{i,j} + U_{i-1,j}}{\Delta x^2} + \frac{U_{i,j+1} - 2\,U_{i,j} + U_{i,j-1}}{\Delta y^2} \right) - \left(V_{1\,i,j} \left(\frac{U_{i+1,j} - U_{i-1,j}}{2\,\Delta x} \right) + V_{2i,j} \left(\frac{U_{i,j+1} - U_{i,j-1}}{2\,\Delta y} \right) \right) + H_{pc}(U_{i,j}) B_{i,j} \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right) - \alpha U_{i,j},$$

and,

$$\dot{B}_{i,j} = -H(U_{i,j})\,\frac{\varepsilon}{q} B_{i,j} \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right).$$

The stability analysis requires the Jacobian Matrix of (27), that is, the computation of the derivative of each vector element with respect to $U_{i,j}$ and $B_{i,j}$, $i = 0, \ldots, N_x, j = 0, \ldots, N_y$ as follows,

$$\mathbf{J}_\Phi = \begin{pmatrix} \frac{\partial \dot{U}_{0,0}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{U}_{0,0}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{U}_{0,0}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{U}_{0,0}}{\partial B_{N_x,N_y}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \dot{U}_{i,j}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{U}_{i,j}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{U}_{i,j}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{U}_{i,j}}{\partial B_{N_x,N_y}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \dot{U}_{N_x,N_y}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{U}_{N_x,N_y}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{U}_{N_x,N_y}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{U}_{N_x,N_y}}{\partial B_{N_x,N_y}} \\ \frac{\partial \dot{B}_{0,0}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{B}_{0,0}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{B}_{0,0}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{B}_{0,0}}{\partial B_{N_x,N_y}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \dot{B}_{i,j}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{B}_{i,j}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{B}_{i,j}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{B}_{i,j}}{\partial B_{N_x,N_y}} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial \dot{B}_{N_x,N_y}}{\partial U_{0,0}} & \cdots & \frac{\partial \dot{B}_{N_x,N_y}}{\partial U_{N_x,N_y}} & \frac{\partial \dot{B}_{N_x,N_y}}{\partial B_{0,0}} & \cdots & \frac{\partial \dot{B}_{N_x,N_y}}{\partial B_{N_x,N_y}} \end{pmatrix}. \tag{28}$$

Then, the elements of (28) are:

$$\frac{\partial \dot{U}_{i,j}}{\partial U_{i,j}} = -2\kappa \left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right) + B_{i,j} \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right) \left(\frac{\partial H_{pc}(U_{i,j})}{\partial U_{i,j}} + \frac{H_{pc}(U_{i,j})}{(1 + \varepsilon U_{i,j})^2} \right) - \alpha, \tag{29}$$

$$\frac{\partial \dot{U}_{i,j}}{\partial U_{i+1,j}} = \frac{\kappa}{\Delta x^2} - \frac{V_{1\,i,j}}{2\Delta x} \tag{30}$$

$$\frac{\partial \dot{U}_{i,j}}{\partial U_{i-1,j}} = \frac{\kappa}{\Delta x^2} + \frac{V_{1\,i,j}}{2\Delta x} \tag{31}$$

$$\frac{\partial \dot{U}_{i,j}}{\partial U_{i,j+1}} = \frac{\kappa}{\Delta y^2} - \frac{V_{2i,j}}{2\Delta y} \tag{32}$$

$$\frac{\partial \dot{U}_{i,j}}{\partial U_{i,j-1}} = \frac{\kappa}{\Delta y^2} + \frac{V_{2i,j}}{2\Delta y} \tag{33}$$

$$\frac{\partial \dot{U}_{i,j}}{\partial B_{i,j}} = H_{pc}(U_{i,j}) \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right) \tag{34}$$

$$\frac{\partial \dot{B}_{i,j}}{\partial U_{i,j}} = -\frac{\varepsilon}{q}\, B_{i,j}\, \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right) \left(\frac{\partial H_{pc}(U_{i,j})}{\partial U_{i,j}} + \frac{H_{pc}(U_{i,j})}{(1 + \varepsilon U_{i,j})^2} \right) \tag{35}$$

$$\frac{\partial \dot{B}_{i,j}}{\partial B_{i,j}} = -H_{pc}(U_{i,j})\, \frac{\varepsilon}{q}\, \exp\left(\frac{U_{i,j}}{1 + \varepsilon U_{i,j}} \right) \tag{36}$$

Notice that $H_{pc}(U_{i,j})$ is not differentiable at u_{pc} but $\frac{\partial H_{pc}(U_{i,j})}{\partial U_{i,j}} = 0$ for $U_{i,j} \neq u_{pc}$. Finally, $\mathbf{J}_\Phi$ is defined by

$$\mathbf{J}_\Phi = \begin{bmatrix} \mathbf{J}_A & \mathbf{J}_B \\ \mathbf{J}_C & \mathbf{J}_D \end{bmatrix} \in \mathbb{R}^{2\,(N_x+1)(N_y+1) \times 2\,(N_x+1)(N_y+1)}, \tag{37}$$

where,

$$\mathbf{J}_A = \begin{bmatrix} \ddots & \ddots & \ddots & & \ddots & \ddots & \ddots & & \ddots & \ddots & \\ \cdots & 0 & (31) & \underbrace{0 \quad \cdots \quad 0}_{N_y \text{ times}} & (33) & (29) & (32) & \underbrace{0 \quad \cdots \quad 0}_{N_y \text{ times}} & (30) & 0 & \cdots \\ \ddots & \ddots & \ddots & & \ddots & \ddots & \ddots & & \ddots & \ddots & \end{bmatrix}, \tag{38}$$

$$\mathbf{J}_B = \begin{bmatrix} (34) & & 0 \\ & \ddots & \\ 0 & & \end{bmatrix}, \quad \mathbf{J}_C = \begin{bmatrix} (35) & & 0 \\ & \ddots & \\ 0 & & \end{bmatrix}, \quad \mathbf{J}_D = \begin{bmatrix} (36) & & 0 \\ & \ddots & \\ 0 & & \end{bmatrix}. \tag{39}$$

Now, the stability analysis requires the computation of the eigenvalues λ of $\mathbf{J}_\Phi(t_n, \mathbf{y}_n)$ for $n = 0, ..., N_t$, and they must be inside the stability region of the *IVP* solver. Let $\bar{z} = \lambda \Delta t \in \mathbb{C}$, the *Euler method* is stable if $|1 + \bar{z}| \leq 1$ and *RK4* is stable if $|1 + \bar{z} + \frac{1}{2}\bar{z}^2 + \frac{1}{6}\bar{z}^3 + \frac{1}{24}\bar{z}^4| \leq 1$. This implies that the algorithm stability depends on Δt selection and the sub-subsection 4.1.1 presents results about experiments of this analysis. Notice that the stability region of *RK4* is larger than the *Euler Method*, allowing the use of larger values of Δt.

3.3 Numerical Implementation

The implementation of the Algorithms 1, 2, and 3 was developed using *Python*. Two libraries were used to performed both implementations for *CPU* and *GPU*.

The first one using *NumPy*, which performs optimized operations over the *RAM* by means the use of the multithreading framework *OpenMP*. Notice that this library allows the storage of vector data structures, and it works as a wrapper of all the necessary linear algebra algorithms implemented in *C/C++* [37].

The second implementation was developed using *CuPy*, an extension of *NumPy* but works with *GPU*, specifically, by means the use of *CUDA* libraries [38]. A *CUDA* compatible *GPU* is required for its operation due to the data manipulation is performed over the *video RAM (vRAM)*. It has almost a full compatibility with the functions used in *NumPy*, so, a very simple way to describe its use is changing the routine from `numpy.routine` to `cupy.routine`.

For both implementations, the spatial partial derivatives in Φ are computed using array slices, which are optimized to achieve short execution times. The original framework described in [6] also includes the matrix-matrix product using numpy.dot, but due to the limited memory for *GPU* implementation, the slice approach was selected since we do not need to storage the differentiation matrices. To solve the *IVP* problem (20), the *Euler Method* and *RK4* method were implemented from scratch using for loops.

It is important to remark that once having the first implementation in *NumPy*, the extension to *CuPy* is almost straightforward by making the changes mentioned above. This allows to obtain a *GPU* implementation with a higher performance than the *CPU* with very little programming effort.

4 Results

4.1 Algorithm Analysis

To evaluate the properties of the algorithm as a valid tool for approximation, this section presents a brief description of the convergence and complexity, and a deep detail the stability analysis of the numerical method. All the analyses elaborated in this section were developed with *Python CPU* implementation.

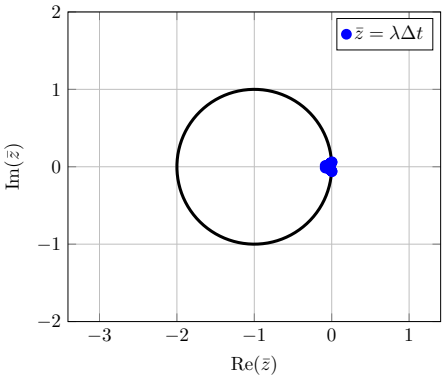
4.1.1 Stability

Stability analysis is performed using $N_x = N_y = 63$, $t_{\max} = 25$, $N_t = 500$ for the *Euler Method*, and $N_t = 100$ for *RK4*. Figure 2 presents the stability region of the *Euler Method* and eigenvalues of $\mathbf{J}_\Phi$ times Δt for the experiment performed. For the *Euler Method*, except in some specific cases for intermediate time-steps, the $\bar{z}_i$ values are slightly outside the stability region when $\Re(\bar{z}_i) < 0$. Although, this did not produce an unstable numerical simulation, we strongly suggests to use *RK4* to avoid any numerical concern.

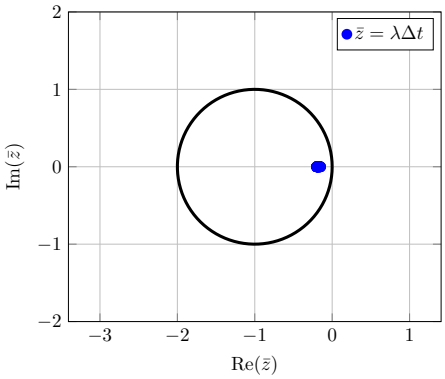
Analysis of experiment for *RK4* is presented in Figure 3. It contains the stability region of the method and eigenvalues of $\mathbf{J}_\Phi$ times Δt.

Unlike the Euler method, in this case the $\bar{z}_i$ are within the stability region for all intermediate time steps.

According to sub-subsection 3.2.5, almost all $\bar{z}_i$ computed for *Euler Method* are inside its stability region but for *RK4* all these values are inside the stability region. These experimental results enable the proposed algorithm as a valid tool for numerically solving the model (1), but it is convenient the use of *RK4* over the *Euler method*.

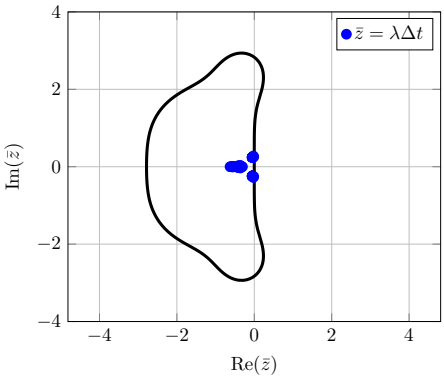


(a) $\bar{z}_i$ for first time step.

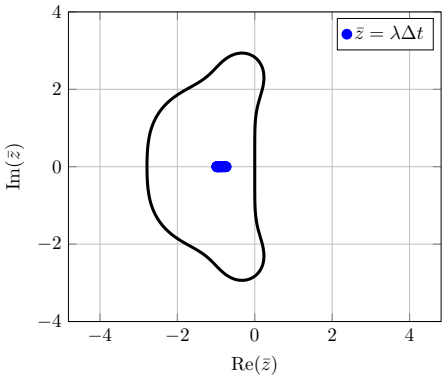


(b) $\bar{z}_i$ for the last time step.

Figure 2: Stability analysis of the algorithm using *Euler* method.



(a) $\bar{z}_i$ for first time step.

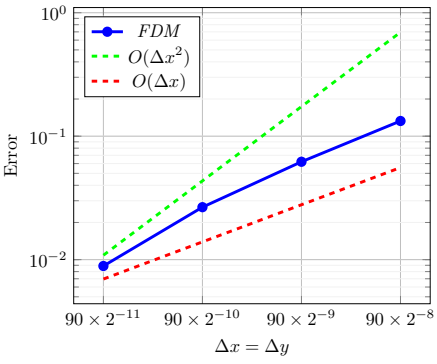


(b) $\bar{z}_i$ for the last time step.

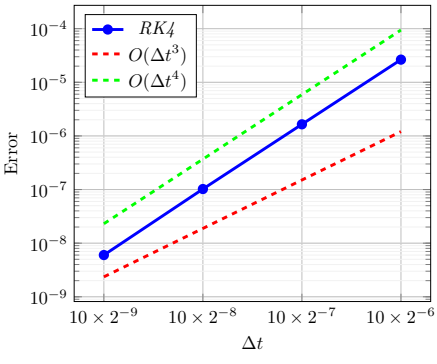
Figure 3: Stability analysis of the algorithm using *RK4* method.

4.1.2 Convergence and Complexity

A convergence and complexity analysis of the algorithm is described in details in [6]. The curves of convergence is presented in Figure 4 and the complexities in Figure 5.



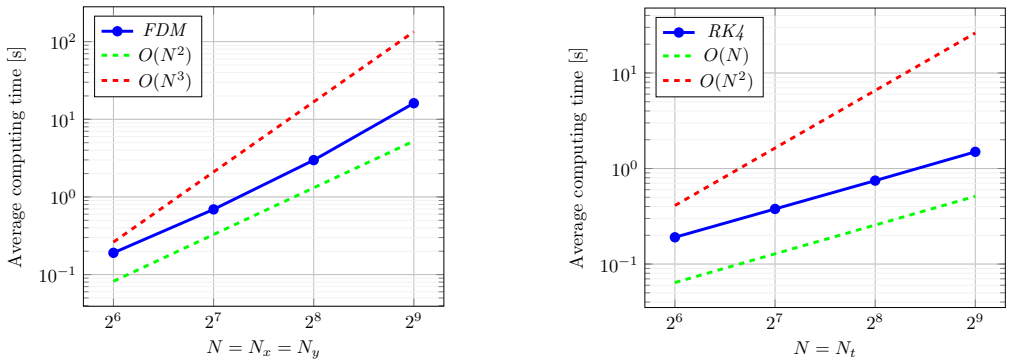
(a) Space convergence.



(b) Time convergence.

Figure 4: Analysis of the algorithm convergence. Based on [6].

These analyses are important because they show that the numerical schemes are consistent because the error for both approximations goes to 0 when $\Delta x = \Delta y \to 0$ and $\Delta t \to 0$, and also the complexity guarantees that the numerical algorithm can compute numerical simulations in affordable times because it is polynomial with respect to the problem size.



(a) Computational complexity of Φ approximation. (b) Computational complexity of the *RK4* method.

Figure 5: Analysis of the algorithm temporal complexity. Based on [6].

4.2 Numerical Simulation

The numerical experiment presented in Figure 6 is performed for a qualitative fire spreading analysis. It tries to reproduce qualitatively the output shown in [18]. The high-temperature zone, or fire front, is formed following the wind direction. Also, the fuel is consumed in the same shape as the fire front. The maximum temperature value reached is around $u = 9$, which in physical variables corresponds to $T = 1110$ [K]. This result is within the values measured in a forest fire and and also agrees with the output reported by [18].

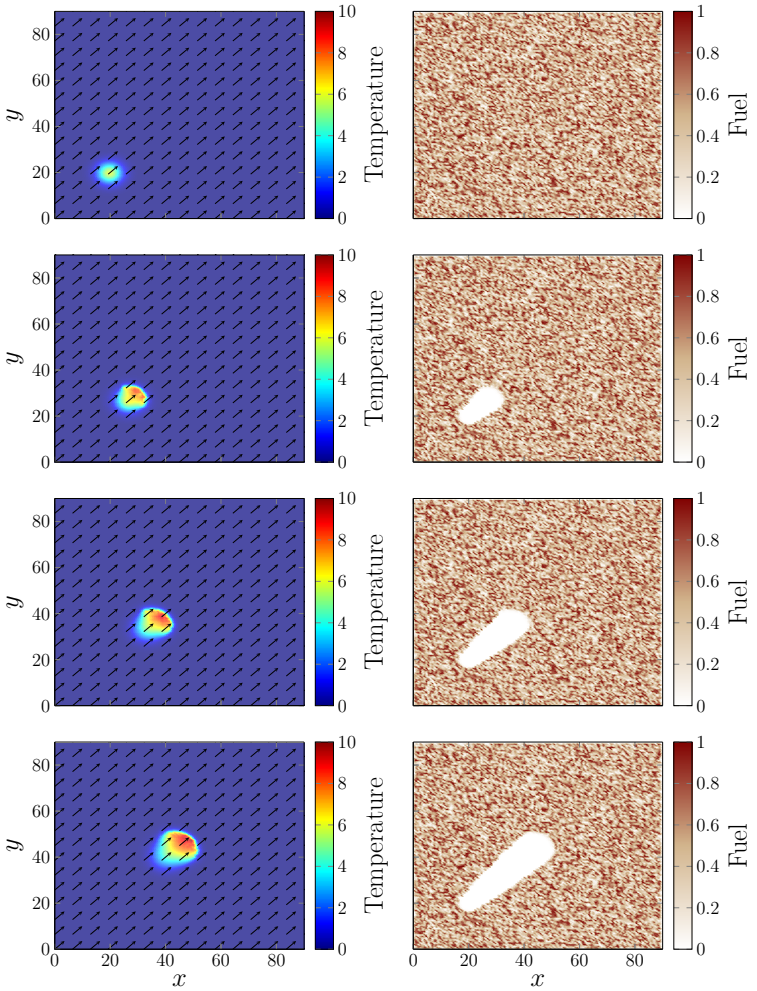


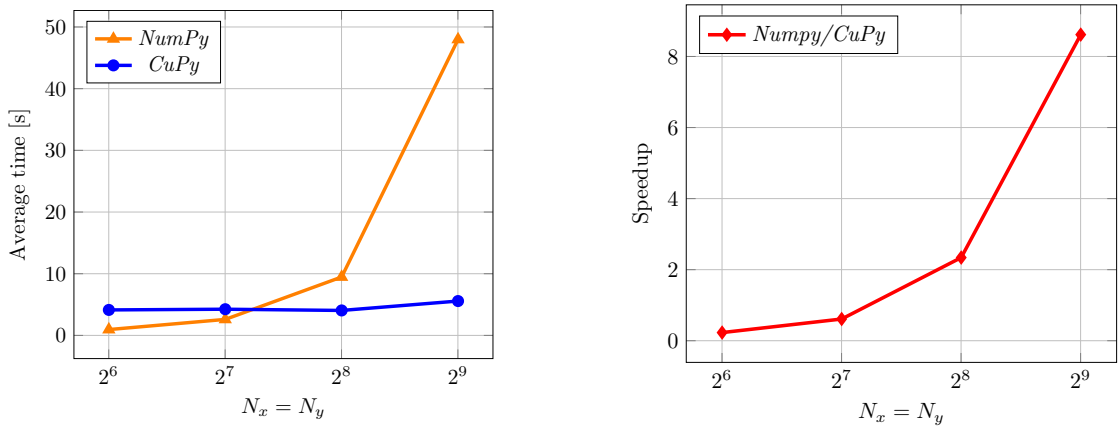
Figure 6: Numerical simulation to achieve [18] output. Left column presents the nondimensional temperature and right column presents the fuel fraction. Times for each row are: $t = \{0, 10, 15, 20\}$. This figure is taken from [6].

4.3 Implementations performance

In order to compare the computational capacity of each of both implementations, we performed 4 simulations for grids size of $N_x = N_y = \{2^6, 2^7, 2^8, 2^9\}$. The idea is to show the capabilities of the *GPU* implementation, but taking care of the limited *vRAM* availability.

The codes were executed in a workstation provided by the *Scientific Computing Team (SCT)* of the Universidad Técnica Federico Santa María. This computer has an Intel(R) Core(TM) i9-10900 *CPU* with 2.80 GHz frequency, 10 physical cores, 20 logical cores and 64 GB of *RAM*. Also, it includes a NVIDIA GeForce RTX 3060 graphic card, which has 3584 cores with 1.32 GHz frequency and 12 GB of *vRAM*.

Figure 7 presents the result of the experiments for the *Numpy* and *CuPy* implementations.



(a) Execution average time of implementations.

(b) Speedup.

Figure 7: *NumPy-CuPy* performance comparison.

From Figure 7a we can figure out that *CuPy* has slower times for experiments of $N \geq 2^8$. It is very important when using *Finite Differences* due to the convergence of the method, requiring a finer mesh for better results. Additionally, Figure 7b shows that for $N = 2^9$ the *CuPy* implementation can be executed more than 8 times faster comparatively with *NumPy* just changing the name of the library in the code. However, it is important to mention that the *GPU* version will be limited according to the graphics card resources available to execute the code.

5 Discussion

Compared to the related work presented in Section 2, this article presents an implementation using a similar model than [18], [19], and [20], but solved using *FDM* for spatial discretization and *RK4* for time integration. Table 2 shows a comparison of the numerical methods used by the related work presented in section 2 and this work. The main advantage of *FDM* is the simplicity of the mathematical formulation versus the *FEM*

Table 2: Related work numerical schemes comparison.

	Discretization	
Model	**Space**	**Time**
Asensio & Ferragut	Finite Element	Semi-implicit Euler
Mandel et al.	Finite Difference	Euler
Eberle et al.	RBF Collocation	Crank-Nicolson
This work	*Finite Difference*	*Fourth-order Runge-Kutta*

or a *RBF* collocation method, but it requires finer meshes for complex geometries. As disadvantage of the *FEM*, it requires a special data structure to handle the computation in addition to the pre-processing related to the computer grid generation. For *RBF*, it depends on the radial basis function used, in addition to the selected parameters according to the problem nature. In terms of implementation *FDM* is easier than the others methods, and also is suitable for the data structure provided by *NumPy* and *CuPy*. On the other hand, for time integration, both semi-implicit Euler and Crank-Nicolson are more stables than explicit methods such as *Euler* or *RK4* methods allowing the use of larger Δt, but they require solving a system

of equations for each time step, which may result in longer computation times. Although this work uses a method that requires evaluating 4 times the right-hand side of the *PDE*, it allows to achieve a better order of convergence.

6 Conclusions

This work has successfully presented two *Python* implementations to support wildfire spreading and effects analysis. Using the numerical implementation of a two-dimensional simplified wildfire model, this work allows the numerical simulation of several fire scenarios. These simulations depend on components such as temperature, fuel fraction, wind, and topography which are parts of the mathematical developed by [18], using the assumptions of [20] . For the numerical approximation of the mathematical model, the proposed algorithm is based on *Method of Lines*, which splits the space and time dependency allowing to approximate independently using different classes of numerical methods. In this implementation was used *FDM* for space and *RK4* for time. Regarding the contribution of this work, this article:

1. Provided analysis and description of the numerical model used.
2. Presented an algorithm with the numerical method used, the convergence, temporal complexity, stability and simulation capabilities. Since the results were positive, we can experimentally assure that the proposed algorithm is valid for the approximation of the mathematical model.
3. Extended a first *NumPy* implementation to *CuPy*, in order to take advantage of the *GPU* performance.
4. Showed the performance and comparison of both *CPU* and *GPU* implementations.

It is important to point out that the *CuPy* implementation allows a fast *GPU* prototype, improving the performance compared to a *CPU* implementation without the necessary effort of directly using frameworks such as *CUDA* or *OpenCL*. This performance improvement is only valid for larger problems mainly because of the fixed cost required in moving memory from *RAM* to *vRAM*.

Satisfactorily, this work goes in the direction to keep improving the current open-source computational tools for wildfire spreading analysis. As results, the developed framework is publicly available in `https://github.com/dsanmartin/ngen-kutral`.

6.1 Future Work

Following the guidelines developed in this work, as future work at least three possible routes can be detached:

1. Explore the use of spectral methods in the approximation of partial derivatives with respect to spatial variables. For instance, the use of Chebyshev's differentiation matrices [35], which require a smaller number of nodes to obtain better approximations when computing the derivatives. Also, the use of the *Fourier Transform* can be explored, taking advantage of the optimized libraries for the computation of the *Fast Fourier Transform* in both *CPU* and *GPU*. A prototype *CPU* version was developed in [39].
2. Use a *GPU* framework such as *CUDA* or *OpenCL*, in order to improve the performance achieved by *CuPy*. These specialized software allow to take better advantage of the resources provided by the *GPU*, due to the memory hierarchy model. In particular, the use of shared memory can significantly accelerate the performance of some operations.
3. Improve the quality of the mathematical model used, coupling a weather model that allows a more realistic wind behavior. For example, the use of the Navier-Stokes equations or some approximation of it that enables the consideration of the atmospheric variation generated by the increase of temperature in a wildfire episode. It is necessary to consider that the numerical methods involved in fluid dynamics are computationally expensive, so they require the use of parallel programming techniques and the access to *High Performance Computing (HPC)* resources.

Acknowledgment

This work was partially supported by ANID-Subdirección de Capital Humano/Doctorado Nacional/2019-21191017, ANID PIA/APOYO AFB180002 Centro Científico Tecnológico de Valparaíso - CCTVal, and Programa de Iniciación a la Investigación Científica (PIIC) from Dirección de Postgrado y Programas, Universidad Técnica Federico Santa María, Chile.

Powered@NLHPC: This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02).

References

[1] S. H. Doerr and C. Santín, "Global trends in wildfire and its impacts: perceptions versus realities in a changing world," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 371, pp. 1–10, 6 2016. [Online]. Available: https://doi.org/10.1098/rstb.2015.0345

[2] S. Archibald, C. E. R. Lehmann, J. L. Gómez-Dans, and R. A. Bradstock, "Defining pyromes and global syndromes of fire regimes." *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, pp. 6442–7, 4 2013. [Online]. Available: https://doi.org/10.1073/pnas.1211466110

[3] S. Costafreda-Aumedes, C. Comas, and C. Vega-Garcia, "Human-caused fire occurrence modelling in perspective: A review," *International Journal of Wildland Fire*, vol. 26, pp. 983–998, 2017. [Online]. Available: https://doi.org/10.1071/WF17026

[4] World Wide Fund and Boston Consulting Group, "Fires, forests and the future: a crisis raging out of control?" WWF, Rue Mauverney 28 1196 Gland, Switzerland, Tech. Rep., 2020. [Online]. Available: https://wwf.panda.org/wwf_news/?661151/fires2020report

[5] H. K. Preisler and A. A. Ager, "Forest-Fire Models," Chichester, UK, 2001. [Online]. Available: https://doi.org/10.1002/9780470057339.vaf010.pub2

[6] D. San Martín and C. E. Torres, "Ngen-Kütral: Toward an Open Source Framework for Chilean Wildfire Spreading," in *2018 37th International Conference of the Chilean Computer Science Society (SCCC)*, 2018, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SCCC.2018.8705159

[7] E. Pastor, L. Zárate, E. Planas, and J. Arnaldos, "Mathematical models and calculation systems for the study of wildland fire behaviour," *Progress in Energy and Combustion Science*, vol. 29, no. 2, pp. 139–153, 1 2003. [Online]. Available: https://doi.org/10.1016/S0360-1285(03)00017-0

[8] F. A. Albini, "Wildland fire spread by radiation-a model including fuel cooling by natural convection," *Combustion Science and Technology*, vol. 45, pp. 101–113, 2 1986. [Online]. Available: https://doi.org/10.1080/00102208608923844

[9] R. O. Weber, "Toward a comprehensive wildfire spread model," *International Journal of Wildland Fire*, vol. 1, no. 4, pp. 245–248, 1991. [Online]. Available: https://doi.org/10.1071/WF9910245

[10] A. M. Grishin, "General mathematical model for forest fires and its applications," *Combustion, Explosion and Shock Waves*, vol. 32, no. 5, pp. 503–519, 1996. [Online]. Available: https://doi.org/10.1007/BF01998573

[11] R. Linn, "A transport model for prediction of wildfire behavior," Ph.D. dissertation, Los Alamos National Laboratory (LANL), Los Alamos, NM, 7 1997. [Online]. Available: https://doi.org/10.2172/505313

[12] R. Linn, J. Reisner, J. J. Colman, and J. Winterkamp, "Studying wildfire behavior using FIRETEC," *International Journal of Wildland Fire*, vol. 11, no. 3-4, pp. 233–246, 2002. [Online]. Available: https://doi.org/10.1071/WF02007

[13] W. Mell, M. A. Jenkins, J. Gould, and P. Cheney, "A physics-based approach to modelling grassland fires," *International Journal of Wildland Fire*, vol. 16, no. 1, pp. 1–22, 2007. [Online]. Available: https://doi.org/10.1071/WF06002

[14] K. McGrattan, R. McDermott, J. Floyd, S. Hostikka, G. Forney, and H. Baum, "Computational fluid dynamics modelling of fire," *International Journal of Computational Fluid Dynamics*, vol. 26, pp. 349–361, 7 2012. [Online]. Available: https://doi.org/10.1080/10618562.2012.659663

[15] M. A. Finney, "Farsite: Fire area simulator-model development and evaluation," pp. 1–47, 1998, fARSITE. [Online]. Available: https://doi.org/10.2737/RMRS-RP-4

[16] C. Lautenberger, "Wildland fire modeling with an eulerian level set method and automated calibration," *Fire Safety Journal*, vol. 62, pp. 289–298, 11 2013. [Online]. Available: https://doi.org/10.1016/j.firesaf.2013.08.014

[17] J. Silva, J. Marques, I. Gonçalves, R. Brito, S. Teixeira, J. Teixeira, and F. Alvelos, "A Systematic Review and Bibliometric Analysis of Wildland Fire Behavior Modeling," *Fluids 2022, Vol. 7, Page 374*, vol. 7, no. 12, p. 374, 12 2022. [Online]. Available: https://doi.org/10.3390/fluids7120374

[18] M. I. Asensio and L. Ferragut, "On a wildland fire model with radiation," *International Journal for Numerical Methods in Engineering*, vol. 54, no. 1, pp. 137–157, 5 2002. [Online]. Available: https://doi.org/10.1002/nme.420

[19] J. Mandel, L. S. Bennethum, J. D. Beezley, J. L. Coen, C. C. Douglas, M. Kim, and A. Vodacek, "A wildland fire model with data assimilation," *Mathematics and Computers in Simulation*, vol. 79, no. 3, pp. 584–606, 12 2008. [Online]. Available: https://doi.org/10.1016/j.matcom.2008.03.015

[20] S. Eberle, W. Freeden, and U. Matthes, "Forest fire spreading," in *Handbook of Geomathematics*, W. Freeden, M. Z. Nashed, and T. Sonar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 1349–1385. [Online]. Available: https://doi.org/10.1007/978-3-642-54551-1_70

[21] P. B. Hansen, "Parallel cellular automata: A model program for computational science," *Concurrency: Practice and Experience*, vol. 5, no. 5, pp. 425–448, 8 1993. [Online]. Available: https://doi.org/10.1002/cpe.4330050504

[22] I. Karafyllidis and A. Thanailakis, "A model for predicting forest fire spreading using cellular automata," *Ecological Modelling*, vol. 99, no. 1, pp. 87–97, 6 1997. [Online]. Available: https://doi.org/10.1016/S0304-3800(96)01942-4

[23] J. Quartieri, N. N. E. Mastorakis, G. Iannone, and C. Guarnaccia, "A Cellular Automata Model for Fire Spreading Prediction," pp. 173–179, 2010.

[24] R. M. Almeida and E. E. N. Macau, "Stochastic cellular automata model for wildland fire spread dynamics," *Journal of Physics: Conference Series*, vol. 285, no. 1, p. 12038, 3 2011. [Online]. Available: https://doi.org/10.1088/1742-6596/285/1/012038

[25] T. Ghisu, B. Arca, G. Pellizzaro, and P. Duce, "An Improved Cellular Automata for Wildfire Spread," *Procedia Computer Science*, vol. 51, pp. 2287–2296, 1 2015. [Online]. Available: https://doi.org/10.1016/j.procs.2015.05.388

[26] N. Fernandez-Anez, K. Christensen, and G. Rein, "Two-dimensional model of smouldering combustion using multi-layer cellular automaton: The role of ignition location and direction of airflow," *Fire Safety Journal*, vol. 91, pp. 243–251, 7 2017. [Online]. Available: https://doi.org/10.1016/j.firesaf.2017.03.009

[27] R. Montenegro, A. Plaza, L. Ferragut, and M. I. Asensio, "Application of a nonlinear evolution model to fire propagation," *Nonlinear Analysis, Theory, Methods and Applications*, vol. 30, no. 5, pp. 2873–2882, 12 1997. [Online]. Available: https://doi.org/10.1016/S0362-546X(97)00341-6

[28] L. Ferragut, M. I. Asensio, and S. Monedero, "Modeling Radiation and Moisture Content in Fire Spread," in *Numerical Mathematics and Advanced Applications*, de Castro Alfredo Bermúdez, , D. Gómez, and Quintela Peregrina, and and Salgado Pilar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 601–608. [Online]. Available: https://doi.org/10.1002/cnm.927

[29] ——, "A numerical method for solving convection-reaction-diffusion multivalued equations in fire spread modelling," *Advances in Engineering Software*, vol. 38, no. 6, pp. 366–371, 6 2007. [Online]. Available: https://doi.org/10.1016/j.advengsoft.2006.09.007

[30] L. Ferragut, M. I. Asensio, J. M. Cascón, and D. Prieto, "A Wildland Fire Physical Model Well Suited to Data Assimilation," *Pure and Applied Geophysics*, vol. 172, no. 1, pp. 121–139, 1 2015. [Online]. Available: https://doi.org/10.1007/s00024-014-0893-9

[31] J. G. Verwer and J. M. Sanz-Serna, "Convergence of method of lines approximations to partial differential equations," *Computing*, vol. 33, no. 3-4, pp. 297–313, 1984. [Online]. Available: https://doi.org/10.1007/BF02242274

[32] E. Sarmin and L. Chudov, "On the stability of the numerical integration of systems of ordinary differential equations arising in the use of the straight line method," *USSR Computational Mathematics and Mathematical Physics*, vol. 3, no. 6, pp. 1537–1543, 1 1963. [Online]. Available: https://doi.org/10.1016/0041-5553(63)90256-8

[33] A. Zafarullah, "Application of the Method of Lines to Parabolic Partial Differential Equations With Error Estimates," *Journal of the ACM*, vol. 17, no. 2, pp. 294–302, 1970. [Online]. Available: https://doi.org/10.1145/321574.321583

[34] S. C. Reddy and L. N. Trefethen, "Stability of the method of lines," *Numerische Mathematik*, vol. 267, pp. 235–267, 1992. [Online]. Available: https://doi.org/10.1007/BF01396228

[35] L. Trefethen, *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 1 2000. [Online]. Available: https://doi.org/10.1137/1.9780898719598

[36] T. Sauer, *Numerical Analysis*, 2nd ed. USA: Addison-Wesley Publishing Company, 2011.

[37] T. E. Oliphant, "Python for scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 10–20, 2007. [Online]. Available: https://doi.org/10.1109/MCSE.2007.58

[38] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "Cupy: A numpy-compatible library for nvidia gpu calculations," in *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. [Online]. Available: https://github.com/cupy/cupy

[39] D. San Martín and C. E. Torres, "Exploring a Spectral Numerical Algorithm for Solving a Wildfire Mathematical Model," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, 11 2019, pp. 1–7. [Online]. Available: https://doi.org/10.1109/SCCC49216.2019.8966412

Appendix

Basic Notation

This section is included to present the basic notation used in the article.

Symbol	Definition
$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$	Gradient operator.
$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$	Laplace operator.
Ω	Spatial domain.
$\Gamma = \partial\Omega$	Spatial domain boundary.
$\mathbf{x} = (x, y)$	Spatial variable.
t	Time variable. As subscript, denotes the time derivative.
κ	Diffusion coefficient.
ε	Inverse of activation energy.
α	Natural convection.
q	Reaction heat.
u_{pc}	Phase change threshold.
D_{N_x}	Differentiation matrix for approximation of $\frac{\partial}{\partial x}$
D_{N_y}	Differentiation matrix for approximation of $\frac{\partial}{\partial y}$
$D^{(2)}_{N_x}$	Differentiation matrix for approximation of $\frac{\partial^2}{\partial x^2}$
$D^{(2)}_{N_y}$	Differentiation matrix for approximation of $\frac{\partial^2}{\partial y^2}$
$\top$	Matrix/vector transpose.
vec	Vectorization operator.
vec^{-1}	Inverse vectorization operator.

Finite Difference formulas

This section presents the Finite Difference formulas used to build the differentiation matrices used by this work. For the first partial derivative with respect to x,

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + O(\Delta x^2), \quad i = 1, \ldots, N_x - 1, j = 0, \ldots, N_y, \tag{40}$$

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{-3\,u_{i,j} + 4\,u_{i+1,j} - u_{i+2,j}}{2\Delta x} + O(\Delta x^2), \quad i = 0, j = 0, \ldots, N_y, \tag{41}$$

$$\left.\frac{\partial u}{\partial x}\right|_{i,j} = \frac{u_{i-2,j} - 4\,u_{i-1,j} + 3\,u_{i,j}}{2\Delta x} + O(\Delta x^2), \quad i = N_x, j = 0, \ldots, N_y. \tag{42}$$

For the second partial derivative with respect to x,

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} = \frac{u_{i+1,j} - 2\,u_{i,j} + u_{i-1,j}}{\Delta x^2} + O(\Delta x^2), \quad i = 1, \ldots, N_x - 1, j = 0, \ldots, N_y, \tag{43}$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} = \frac{2\,u_{i,j} - 5\,u_{i+1,j} + 4\,u_{i+2,j} - u_{i+3,j}}{\Delta x^2} + O(\Delta x^2), \quad i = 0, j = 0, \ldots, N_y, \tag{44}$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} = \frac{-u_{i-3,j} + 4\,u_{i-2,j} - 5\,u_{i-1,j} + 2\,u_{i,j}}{\Delta x^2} + O(\Delta x^2), \quad i = N_x, j = 0, \ldots, N_y. \tag{45}$$

For the first partial derivative with respect to y:

$$\left.\frac{\partial u}{\partial y}\right|_{i,j} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = 1, \ldots, N_y - 1, \tag{46}$$

$$\left.\frac{\partial u}{\partial y}\right|_{i,j} = \frac{-3\,u_{i,j} + 4\,u_{i,j+1} - u_{i,j+2}}{2\Delta y} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = 0, \tag{47}$$

$$\left.\frac{\partial u}{\partial y}\right|_{i,j} = \frac{u_{i,j-2} - 4\,u_{i,j-1} + 3\,u_{i,j}}{2\Delta y} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = N_y. \tag{48}$$

For the second partial derivative with respect to y:

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} = \frac{u_{i,j+1} - 2\,u_{i,j} + u_{i,j-1}}{\Delta y^2} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = 1, \ldots, N_y - 1, \tag{49}$$

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} = \frac{2\,u_{i,j} - 5\,u_{i,j+1} + 4\,u_{i,j+2} - u_{i,j+3}}{\Delta y^2} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = 0, \tag{50}$$

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} = \frac{-u_{i,j-3} + 4\,u_{i,j-2} - 5\,u_{i,j-1} + 2\,u_{i,j}}{\Delta y^2} + O(\Delta y^2), \quad i = 0, \ldots, N_x, j = N_y. \tag{51}$$

Vectorization

We included this section to define the vectorization operator vec and the inverse vec^{-1}. The vectorization operator used by this work is defined in column-major order as follows. Let $A \in \mathbb{R}^{m \times n}$,

$$A = \begin{pmatrix} a_{1,1} & a_{1,1} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ddots & a_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n-1} & a_{m,n} \end{pmatrix}, \tag{52}$$

then, the vectorization operator is defined by

$$\text{vec}(A) = (a_{1,1}, a_{2,1}, \ldots, a_{m,1}, a_{1,2}, a_{2,2} \ldots, a_{m-1,n}, a_{m,n})^\top = \mathbf{a}. \tag{53}$$

To restore the original shape of a vectorized matrix $\mathbf{a} = \text{vec}(A) \in \mathbb{R}^{mn}$,

$$\text{vec}^{-1}(\mathbf{a}) = \begin{pmatrix} a_{1,1} & a_{1,1} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ddots & a_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n-1} & a_{m,n} \end{pmatrix} = A. \tag{54}$$