TESIS MAESTRÍA EN FÍSICA

FRENTES DE ONDA EN ECUACIONES DE REACCIÓN-DIFUSIÓN-CONVECCIÓN SOBRE MEDIOS HETEROGÉNEOS

Lic. Renzo Zagarra Saez
Maestrando

Dr. Alejandro Kolton
Director

Miembros del Jurado
Dr. Ezequiel Ferrero

3 de Noviembre de 2022

Teoría de la Materia Condensada – Centro Atómico Bariloche

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

A mis padres, Francisco y Liliana,
por su apoyo incondicional.
A mis hermanos, Franco y Lucas,
por hacer de cada momento una sonrisa.
A mi novia, Sol,
por acompañarme y entenderme como nadie.

Índice de contenidos

Índice de símbolos

$S(x, y, t)$ Fracción de susceptibles en la posición (x, y) en el instante t.
$I(x, y, t)$ Fracción de infectados en la posición (x, y) en el instante t.
$R(x, y, t)$ Fracción de recuperados en la posición (x, y) en el instante t.
β Tasa de transmisión.
γ Tasa de recuperación.
S_0 Distribución inicial de la fracción de susceptibles.
S_c Fracción de susceptibles crítica.
R_0 Coeficiente de reproducción.
D_x Coeficiente de dispersión de x.
$u(y, t)$ Campo de desplazamiento del frente de onda.
$u_{cm}(t)$ Centro de masa del frente de onda.
I_{max} Amplitud media del frente de onda.
c Velocidad media del frente de onda.
$w(t)$ Rugosidad del frente de onda.
$S(q)$ Factor de estructura del campo de desplazamiento del frente de onda.
$f_I(t)$ Perfil centrado del frente de onda.
$\beta_{\mathbf{r}}$ Distribución espacial de la tasa de transmisión.
H Medio homogéneo
DA Heterogeneidad dicotómica-aleatoria.
S Heterogeneidad suavizada.
DC Heterogeneidad dicotómica-correlacionada.

Índice de figuras

Índice de tablas

Resumen

Se estudió la propagación de frentes de onda gobernados por ecuaciones de reacción-difusión en el marco del modelo SIR espacial. Dichos frentes de onda podrían utilizarse para caracterizar frentes de infección en una problemática epidemiológica o bien orientarse a una problemática completamente diferente como lo son los frentes de incendios. Se definió una metodología estadística para la caracterización de los frentes de onda a partir de la cual se obtuvieron resultados cuantitativos respecto de la velocidad, la amplitud media, las propiedades geométricas e incluso la nocividad de los frentes sobre diferentes medios isotrópicos. En particular, se exploraron medios homogéneos, desordenados y correlacionados a partir de lo cual pudo describirse cuantitativamente el efecto que tenía cada uno de ellos sobre las características del frente de onda.

Se realizaron simulaciones numéricas masivas para resolver el sistema de ecuaciones de reacción-difusión involucrado en la dinámica. Estas se implementaron de manera eficiente utilizando computación acelerada a través de programación en paralelo sobre procesadores gráficos. De esta manera fue posible obtener resultados sobre sistemas a gran escala en tiempos razonables.

Palabras clave: SISTEMAS COMPLEJOS, MEDIOS DESORDENADOS, ECUACIONES DE DIFUSIÓN, MODELO SIR

Abstract

The propagation of wave fronts governed by reaction-diffusion equations were studied within the framework of the spatial SIR model. These wave fronts could be used to characterize infection fronts in an epidemiological problem or be oriented to a completely different problem such as fire fronts. A statistical methodology was defined for the characterization of the wave fronts from which quantitative results were obtained regarding the speed, the mean amplitude, the geometric properties and even the harmfulness of the fronts on different isotropic media. In particular, homogeneous, disordered and correlated media were explored, from which it was possible to quantitatively describe the effect that each of them had on the characteristics of the wavefront.

Massive numerical simulations were performed to solve the system of reaction-diffusion equations involved in the dynamics. These were efficiently implemented using accelerated computing through parallel programming on graphics processors. In this way it was possible to obtain results on large-scale systems in reasonable times.

Keywords: DYNAMIC SYSTEMS, DISORDERED MEDIA, DIFFUSION EQUATIONS, SIR MODEL

Capítulo 1

Introducción

El modelado matemático de una dada fenomenología constituye una herramienta fundamental en el proceso de entendimiento cuantitativo de la misma. Más aún, aplicado correctamente sobre una problemática concreta, como lo son las epidemias o los incendios forestales, permite desarrollar estrategias de contención, mitigación y prevención [1].

Entre la gran diversidad de desafíos que se presentan al momento de describir la dinámica de enfermedades infecciosas sobre una dada población o bien la propagación de un frente de incendio, se encuentra el desafío de representar correctamente el carácter heterogéneo de la distribución espacial de la población o vegetación [2]. Esto, en última instancia, incluiría aspectos desde el ámbito comportamental de los individuos hasta la distribución espacial de los mismos. En tanto que para incendios forestales, involucra la topografía del terreno, la diversidad de vegetación y su distribución espacial y hasta contribuciones climáticas.

El objetivo de este trabajo de tesis es dar un paso en esta dirección. Tanto para comprender los efectos que tienen sobre la dinámica las heterogeneidades del medio de sustentación, ya sea de la población o vegetación, como para desarrollar herramientas estadísticas y computacionales que puedan ser utilizadas en sistemas completamente diferentes donde la influencia de las características del medio sean de inteŕes. Para ello se consideró un modelo espacio-temporal de los más sencillos en lo que respecta a modelos epidemiológicos de tipo SIR (Susceptibles - Infectados - Recuperados) [3–6], en donde la movilidad de los individuos es dominada por un término difusivo[7], tal como se verá en detalle en el capítulo **??**.

Por su parte, las heterogeneidades del medio se introdujeron por medio de la distribución espacial de la tasa de transmisión la cual ha mostrado tener implicaciones significativas para reproducir patrones de propagación espacio-temporales dados por datos epidemiológicos [8].

El presente trabajo se divide en tres capítulos además del presente, los cuales se describen brevemente a continuación:

En el **Capítulo 2** se presenta el marco teórico del trabajo, las herramientas estadísticas utilizadas para caracterizar los frentes de onda y se precisa las condiciones del problema a resolver junto con la caracterización de los distintos medios que se propone estudiar.

En el **Capítulo 3** se presentan los resultados obtenidos a partir de las simulaciones numéricas realizadas masivamente para cubrir diferentes parámetros del problema y fundamentalmente para cuantificar los efectos sobre la dinámica de los distintos medios de sustentación.

En el **Capítulo 4** se comentan brevemente las conclusiones del trabajo, sus potenciales aplicaciones y un posible desarrollo a futuro.

Capítulo 2

Modelo SIR

2.1. Historia

Desde tiempos remotos las epidemias y pandemias han sido origen de sufrimiento para la humanidad. Cada individuo, pueblo, ciudad o civilización amenazada por una epidemia ha tenido que enfrentarlas de una u otra manera. La historia de la humanidad está llena de acontecimientos epidemiológicos que han causado grandes pérdidas de vidas humanas y cambios en la forma de vida de las sociedades. La peste negra, la gripe española, la viruela, la tuberculosis, la malaria y la fiebre amarilla, entre otras, han sido algunas de las epidemias más devastadoras de la historia.

Peste Negra

En la Edad Media, la peste negra, que se extendió por Europa entre 1346 y 1353, causó la muerte de entre un tercio y la mitad de la población europea. La enfermedad se transmitía por medio de la picadura de pulgas infectadas que vivían en las ratas. De hecho, el mecanismo de transmisión de la enfermedad fue un misterio durante mucho tiempo. En 1855, una epidemia de peste bubónica comenzó en el sur de China. Fue entonces cuando pudieron obtenerse resultados de laboratorio y recién en 1897, M. Ogata, y en 1898, P. L. Simonds, de manera independiente llegaron a la hipótesis de que la enfermedad se transmitía por medio de la picadura de pulgas infectadas. Lo cual fue confirmado unos años más tarde.

En resumen, el mecanismo de transmisión era el siguiente: algunas de las pulgas sufrían un bloqueo estomacal luego de alimentarse de una rata infectada, como consecuencia, estas pulgas comenzaban a picar con mayor frecuencia de lo habitual dado que la sangre que ingerían no les llegaba a los intestinos para ser digerida. El punto crucial de la cadena de transmisión, reside en la existencia de dos poblaciones distintas de ratas, una resistente y otra que no lo es. La especie resistente era la responsable de mantener la enfermedad endémica albergando las pulgas infectadas. Mientras que la especia no resistente generaba una escasez de alimento para las pulgas, las cuales se veían forzadas a cambiar de huésped, en este caso, el humano. Finalmente, la enfermedad se transmitía cuando las pulgas con bloqueo estomacal vomitaban sangre infectada al picar a una persona [10].

La plaga se introdujo a Europa por Italia en diciembre de 1347 y se extendió rápidamente, a un ritmo de 300 - 600 km por año [11]. En la figura 2.1 se muestra la propagación espacio-temporal de la peste negra en Europa. Las rutas comerciales fueron las principales vías de propagación de la plaga.

Gripe Española

La gripe española, por su parte, se extendió por todo el mundo entre 1918 y 1920, se estima que causó la muerte de una 40 millones de personas y entre un 25-30 % de la población mundial padeció la enfermedad [12]. Se originó en Kansas, Estados Unidos, y se extendió por todo el mundo en menos de un año. Curiosamente, se denominó gripe española simplemente porque España no participó en la Primera Guerra Mundial que acontecía simultáneamente, por lo tanto, no se censuraron los datos de la enfermedad. La gripe española fue una pandemia de gripe de tipo A,

Figura 2.1: Mapa de la peste negra en Europa [9].

causada por el virus H1N1. Fue la primera de tres pandemias causadas por este virus, la última de las cuales ocurrió en 2009.

A diferencia de otros virus de tipo A, que típicamente tienen mayor mortalidad sobre niños y ancianos, este virus impactó fuertemente sobre la población joven-adulta (20-40 años) [13]. El mecanismo de transmisión es como el de la gripe común, es decir, aéreo, por medio de gotitas de saliva que se expulsan al toser o estornudar.

Es interesante notar en este caso cómo el contexto histórico influyó en la propagación de la enfermedad. En 1918, la Primera Guerra Mundial estaba en su apogeo y muchos países se encontraban en medio de una crisis económica. Se piensa que la elevada tasa de transmisión de la enfermedad, con un número de reproducción básico entre 2 y 3 [14], se debió a la enorme cantidad de tropas movilizadas por todo el mundo. Además, esto mismo sustentó mutaciones en el virus.

2.2. Modelo SIR de campo medio

El modelo SIR describe la dinámica de tres grupos característicos de un sistema, denominados comúnmente como **S**usceptibles, **I**nfectados y **R**ecuperados, de ahí su nombre. Es el modelo más simple para describir la propagación de una enfermedad infecciosa, fue propuesto originalmente hace casi un siglo, en 1927, por *Kermack* y *McKendrick* [3].

El mismo está formulado de la siguiente manera: sean $S(t)$, $I(t)$ y $R(t)$ la fracción de susceptibles, infectados y recuperados de una población dada a tiempo t respectivamente, entonces, la dinámica de estos grupos está descrita por el siguiente sistema de ecuaciones diferenciales ordinarias

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\beta SI, \tag{2.1}$$

$$\frac{\mathrm{d}I}{\mathrm{d}t} = \beta SI - \gamma I, \tag{2.2}$$

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \gamma I. \tag{2.3}$$

Donde β corresponde a una tasa de transmisión mientras que γ corresponde a una tasa de recuperación.

Usualmente, la ecuación para R (2.3) no se escribe, ya que puede ser reemplazada por la más simple $S+I+R=1$. Puede verse por inspección que el sistema de ecuaciones (2.1 - 2.3) cumple $\frac{\mathrm{d}S}{\mathrm{d}t} + \frac{\mathrm{d}I}{\mathrm{d}t} + \frac{\mathrm{d}R}{\mathrm{d}t} = 0$.

Para entender por qué este simple sistema de ecuaciones (2.1 - 2.3) podría describir la dinámica del problema observamos que el mismo está compuesto esencialmente por dos términos, el término de transmisión βSI y el de recuperación γI. Cualitativamente, resulta razonable que la magnitud de sujetos infectados por unidad de tiempo aumente con el producto de la cantidad de infectados y susceptibles, de ahí el término de transmisión. Por otro lado, la cantidad de infectados que se recuperan por unidad de tiempo es entendible que sea proporcional a la misma cantidad de infectados y de ahí el término de recuperación. Por supuesto, es posible justificar esto de una manera más cuantitativa y precisa, para ver una derivación de estas ecuaciones consúltese [4].

A pesar de su simplicidad, este modelo (2.1 - 2.3) no puede resolverse explícitamente. Es decir, no puede hallarse una expresión analítica exacta para $I(t)$ y $S(t)$ que nos permita anticipar la cantidad de infectados que habrá a tiempo t dadas las condiciones iniciales $I(0) = I_0$ y $S(0) = S_0$[1]. Por ello es necesario recurrir a métodos numéricos para resolverlo. En la figura 2.2 se puede ver la evolución temporal de las variables del modelo resuelto numéricamente[2] usando los parámetros $\beta = 5/\text{semana}$ y $\gamma = 1/\text{semana}$ con condiciones iniciales $I_0 = 0.01$ y $S_0 = 0.99$. Se observa cómo la fracción de susceptibles decrece mientras la de infectados aumenta hasta llegar a un pico donde aproximadamente la mitad de la población está infectada. Luego los infectados comienzan a recuperarse y casi toda la población termina en la clase R, de modo que la mayoría de la población atravesó la enfermedad.

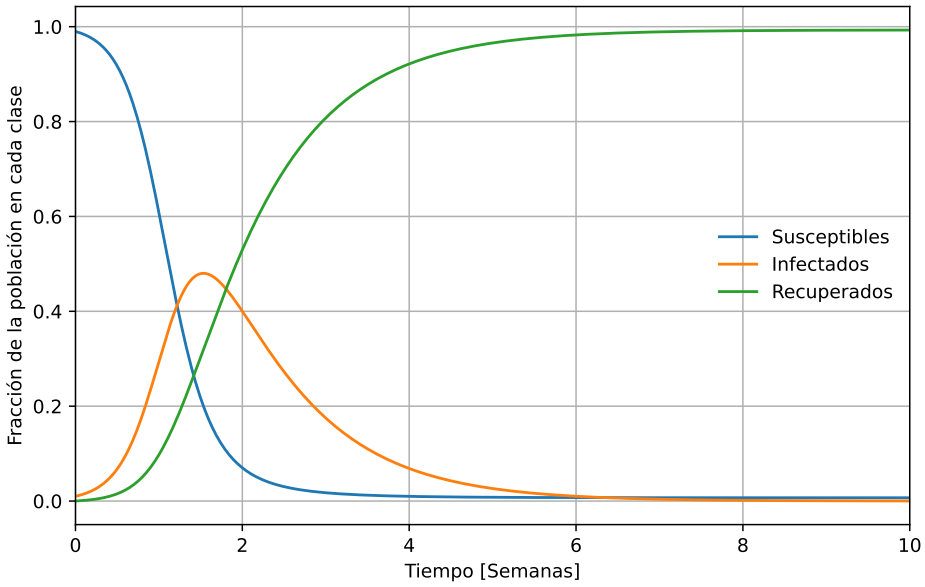

Figura 2.2: Evolución temporal de las variables del modelo resuelto numéricamente con $\beta = 5/\text{semana}$, $\gamma = 1/\text{semana}$, $I_0 = 0.01$ y $S_0 = 0.99$.

Es claro que la figura 2.2 no muestra toda la riqueza del sistema, ya que simplemente muestra la solución para

[1] En realidad, para ser precisos, en 2014 se encontró una solución analítica, en términos de una integral que debe resolverse numéricamente [15].

[2] Se utilizó Runge-Kutta de cuarto orden para la integración numérica [16].

un solo conjunto de parámetros y condiciones iniciales. Es decir, es de esperar que la dinámica difiera si por ejemplo la tasa de transmisión β es menor. En particular, es de interés saber qué conjunto de parámetros β y γ favorecen o dificultan el progreso de la infección. Esto puede determinarse de manera sencilla pidiendo que $\frac{\mathrm{d}I}{\mathrm{d}t} < 0$ al momento del brote de la infección. De esto resulta que si $S_0 < S_c = \gamma/\beta$ para cualquier $I_0 > 0$ entonces la infección no progresa. Este es un resultado conocido, obtenido por Kermack y McKendrick (1927).

El cociente γ/β es la tasa de recuperación relativa, sin embargo, su recíproco le quita todos los méritos, $R_0 = \beta/\gamma$ conocido en epidemiología comúnmente como el número de reproducción básico, el cual describe la media de personas infectadas por un individuo infectado. Dado que usualmente $S_0 \approx 1$, la condición para que la infección perezca se lee ahora en función de R_0 simplemente como $R_0 < 1$. Lo cual resulta natural, si un infectado infecta en promedio a menos de una persona en lo que cursa la enfermedad entonces la infección no se propaga. En la figura 2.3 se puede ver la evolución temporal de las variables del modelo resuelto numéricamente con distintos números de reproducción básico R_0. Se observa que en la medida que $R_0 \to 1$ la magnitud del máximo de infección va decreciendo, mientras que cuando $R_0 = 1$ la cantidad de infectados solo decrece en el tiempo hasta llegar a cero. Lo mismo sucede si $R_0 < 1$.

Es importante no confundir el número de reproducción básico R_0 con el número de reproducción efectivo $R_e(t)$, el primero considera que todos los individuos de la población son susceptibles, mientras que el segundo no y varía con el tiempo, se define como $R_e(t) = \beta S(t)/\gamma$. Este probablemente sea el indicador epidemiológico más usado e importante. De manera similar que con R_0, se puede ver que si $R_e(t) < 1$ entonces la tasa de infección $\frac{\mathrm{d}I}{\mathrm{d}t}(t)$ es decreciente.

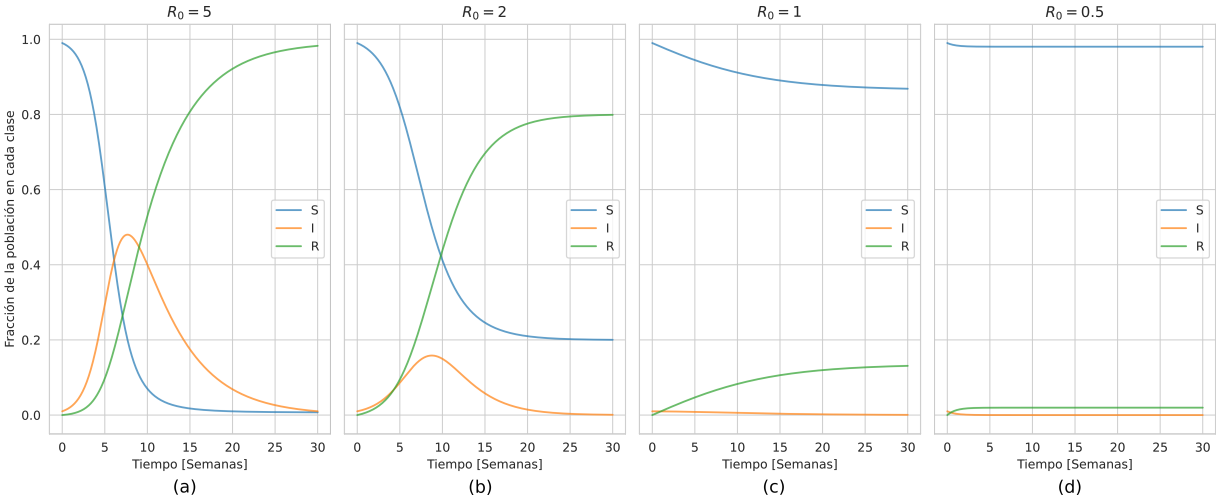

Figura 2.3: Evolución temporal de las variables del modelo resuelto numéricamente con (a) $R_0 = 5$, (b) $R_0 = 2$, (c) $R_0 = 1$ y (d) $R_0 = 0.5$.

Otra cantidad de interés es la fracción de la población que no se contagió nunca. Es decir, cuánto vale el límite $\lim_{t\to\infty} S(t) \equiv S(\infty)$. De la figura 2.3 se puede ver que cuando $R_0 = 5$ la fracción de la población que no se contagia es muy cercana a cero, pero distinta a cero en fin. Mientras que cuando $R_0 = 2$, pareciera que $S(\infty) \to 0.2$. Para responder esta pregunta de manera más general podemos tomar la ecuación 2.1, dividirla por 2.3 y luego integrar para obtener una expresión de S en función de R. De esto resulta que,

$$S(t) = S_0 e^{-R_0 R(t)},$$

tomando el límite correspondiente, usando $\lim_{t\to\infty} R(t) \equiv R(\infty)$ y que $R(\infty) = 1 - S(\infty)$, resulta la siguiente ecuación trascendental para $S(\infty)$

$$\begin{aligned} S(\infty) &= S_0 e^{-R_0(1-S(\infty))}, \\ 0 &= S(\infty) - S_0 e^{-R_0(1-S(\infty))}. \end{aligned} \tag{2.4}$$

Puede verse que si $S(\infty) = S_0$, entonces el miembro derecho de 2.4 es positivo, mientras que si $S(\infty) = 0$, es negativo.

De modo que la ecuación 2.4 tiene una solución para $S(\infty)$ entre 0 y 1. Es decir, sin importar qué tan grande sea el número de reproducción básico R_0, o bien, qué tan contagiosa sea la enfermedad, siempre existe una fracción de la población que no se contagia.

Es importante señalar brevemente las virtudes y fundamentalmente las hipótesis bajo las que se presenta este modelo. La ventaja más notable es la simplicidad y el carácter didáctico del mismo, que como vimos, permite definir y asimilar conceptos generales asociados a la problemática de manera sencilla. Esta simplicidad, sin embargo, viene acompañada de hipótesis que en ocasiones resultan restrictivas y poco realistas en lo que respecta a una dinámica tan compleja como la de una epidemia. Por ejemplo, se ignoran efectos de demografía los cuales pueden tener un impacto apreciable sobre la dinámica a escalas temporales extensas propias de una endemia. Además, se trata de un modelo de campo medio, donde se asume que cada sujeto de la población interactúa con todos los demás, es decir, desprecia heterogeneidades que puedan surgir de la edad, el espacio o aspectos de comportamiento. Adicionalmente, supone que individuos que pasaron por la enfermedad adquieren inmunidad para toda la vida y que un sujeto inmedianamente infectado puede infectar a otro. Por supuesto, esto no desmerece en nada al modelo, el cual sigue siendo extremadamente útil como primera aproximación al modelado de este tipo de sistemas complejos, simplemente es importante recordar las hipótesis sobre las que se trabaja para evitar posibles confusiones.

Por último, es de interés mencionar que si bien el enfoque ha estado hasta ahora centrado en una descripción epidemiológica, es posible extender este mismo modelo de manera sencilla a otras problemáticas. En particular, puede asociarse rápidamente el modelo SIR con la dinámica de un incendio en un bosque. Donde los «susceptibles» son los árboles que pueden incendiarse, los «infectados» son los árboles en llamas y los «recuperados» los árboles que ya han sido quemados y no pueden volver a incendiarse.

2.3. Modelo SIR de reacción-difusión homogéneo

De manera general las ecuaciones de reacción-difusión sobre un medio isotrópico son aquellas que pueden escriberse como

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f} + \boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\mathbf{u}), \tag{2.5}$$

donde $\mathbf{u}$ es un campo vectorial que dependende de la posición $\mathbf{x}$ y el tiempo t, $\mathbf{f}$ es el término de reacción, que es función de $\mathbf{u}$, $\mathbf{x}$ y t, mientras que $\boldsymbol{\nabla} \cdot (D\boldsymbol{\nabla}\mathbf{u})$ es el término de difusión, donde D es la matriz de difusión que puede ser función de $\mathbf{x}$. Este tipo de ecuaciones han sido ampliamente estudiadas [4, 11, 17, 18] y se denominan así porque originalmente se utilizaron para estudiar la dinámica de reactivos químicos.[19]

En lo que respecta al modelo SIR de reacción-difusión que nos interesa a nosotros, este puede escribirse de manera sencilla agregando el término difusivo a las ecuaciones (2.1 - 2.3), dejando de lado la ecuación para R, esto es

$$\partial_t S = -\beta SI + D_S \nabla^2 S, \tag{2.6}$$

$$\partial_t I = \beta SI - \gamma I + D_I \nabla^2 I. \tag{2.7}$$

Donde ahora S, I y R son funciones de la posición (x, y) en un espacio bidimensional además del tiempo t, de modo que ahora se cumple $S(x, y, t) + I(x, y, t) + R(x, y, t) = 1$ para todo (x, y, t). En este nuevo modelo (2.6 - 2.7) los términos difusivos dan lugar a una transmisión local de la infección. Es decir, abandonamos el modelo de campo medio que teníamos en la sección 2.2 donde todos los individuos podían interactuar entre sí.

Hemos supuesto que los coeficientes de difusión D_S y D_I son independientes de la posición y que la matriz D es diagonal dejando de lado la posibilidad de difusión cruzada. Adicionalmente, tanto β como γ son independientes de la posición, dando lugar a un medio totalmente homogéneo, todos los puntos del espacio son equivalentes en términos de transmisión. Esta es una característica crítica a remarcar, ya que en la sección 2.4 presentamos el correpondiente modelo heterogéneo, donde el medio puede adquirir un carácter desordenado, que es el foco de estudio de este trabajo.

A continuación se muestran algunos resultados interesantes asociados a este modelo que serán de interés a la hora de compararlo con su versión heterogénea.

2.3.1. Soluciones de onda

Dada una fracción de infectados inicial $I(x, y, 0)$ y una fracción de susceptibles distribuída homogéneamete $S(x, y, 0) = S_0$, se quiere saber cómo es la evolución espacio-temporal de la fracción de infectados $I(x, y, t)$. En la problemática de incendios la idea sería la misma, pero cambiando infectados por, digamos, incendiados.

Nuevamente, no es posible resolver las ecuaciones 2.6 y 2.7 de manera exacta, sin embargo, es posible estudiar qué condiciones deben satisfacerse para que cierto tipo de soluciones puedan existir. En particular, nos interesa estudiar bajo qué condiciones podría existir una solución de onda y qué características tendría.

Para ello proponemos una solución de onda plana donde

$$S(x, y, t) = S(z), \qquad I(x, y, t) = I(z), \qquad z = x - ct, \tag{2.8}$$

que representa una onda de infección viajando en la dirección x positiva con una velocidad $c > 0$. Reemplazando 2.8 en 2.6 y 2.7, resulta el siguiente sistema de ecuaciones no lineales,

$$D_S S'' + cS' - \beta IS = 0, \tag{2.9}$$

$$D_I I'' + cI' + \beta I(S - \gamma/\beta) = 0, \tag{2.10}$$

donde las primas indican derivadas respecto de z. Como es habitual, este sistema tampoco puede resolverse explícitamente, sin embargo, imponiendo las siguientes condiciones para las soluciones[3]

$$I(\pm\infty) = 0, \qquad S(\infty) = S_0, \qquad S(-\infty) = S_1,$$

donde S_1 sería la fracción de susceptibles que deja la onda por detrás, es posible linearizar la ecuación 2.10 para z donde $S(z) \approx S_0$, es decir, sobre el perfil frontal de la onda. De lo cual resulta,

$$D_I I'' + cI' + \beta I(S_0 - \gamma/\beta) = 0, \tag{2.11}$$

que tiene una solución $I(z) \propto e^{-\lambda z}$, con λ satisfaciendo,

$$D_I \lambda^2 - c\lambda + \beta(S_0 - \gamma/\beta) = 0,$$

es decir,

$$\lambda = \frac{c}{2D_I} \pm \sqrt{(c/2D_I)^2 - \frac{\beta}{D_I}(S_0 - \gamma/\beta)}.$$

Debemos imponer además que $\lambda \in \mathbb{R}$ con $\lambda > 0$, de otra manera la solución no sería autoconsistente. Al imponer que $\lambda > 0$ resulta $\gamma/\beta < S_0$, que es la misma condición que habíamos obtenido en la sección 2.2 para que la infección progrese, mientras que ahora es una condición necesaria para la existencia de soluciones onda, las cuales darían lugar a la propagación de la infección, por lo menos resulta concordante. Más interesante quizás, es la condición $\lambda \in \mathbb{R}$, de la cual resulta que

$$0 \leq (c/2D_I)^2 - \frac{\beta}{D_I}(S_0 - \gamma/\beta),$$

$$c \geq 2\sqrt{D_I \beta (S_0 - \gamma/\beta)} \equiv c_0,$$

[3] Se utiliza la siguiente notación por simplicidad, dada una función $f(z)$,

$$\lim_{z \to \pm\infty} f(z) \equiv f(\pm\infty)$$

dando así una velocidad mínima c_0 para la existencia de la onda. Veremos en la sección **??** que la velocidad de propagación es en realidad muy cercana a la mínima encontrada aquí c_0. Tomando esto por cierto, el perfil frontal de la onda de infección está dado por

$$I_f(z) \propto \exp\left[-\frac{c_0}{2D_I}z\right]. \tag{2.12}$$

Haciendo una cuenta equivalente para el perfil posterior de la onda donde $S(z) \approx S_1$, hay que resolver la ecuación análoga a 2.11, $D_I I'' + cI' + \beta I(S_1 - \gamma/\beta) = 0$, de esto resulta

$$I_p(z) \propto \exp\left[\left(-\frac{c_0}{2D_I} + \sqrt{(c_0/2D_I)^2 - \frac{\beta}{D_I}(S_1 - \gamma/\beta)}\right)z\right], \tag{2.13}$$

$$I_p(z) \propto \exp\left[\left(-\frac{c_0}{2D_I} + \sqrt{\frac{\beta}{D_I}(S_0 - S_1)}\right)z\right]. \tag{2.14}$$

De la ecuación 2.13 se ve que es necesario que $\gamma/\beta > S_1$, dado que debe satisfacerse $I_p(-\infty) = 0$, de modo que en resumen se tiene la siguiente relación

$$S_0 > \gamma/\beta > S_1 > 0,$$

es decir, que la fracción de susceptibles que quedan tras el paso de la onda no es suficiente para activar una onda de retroceso ya que $S_1 < \gamma/\beta$.

En resumen, se obtuvieron expresiones analíticas aproximadas del perfil frontal y posterior que tendría una solución de onda, ecuaciones 2.12 y 2.13 respectivamente, las cuales dan a entender que el perfil completo de la onda es asimétrico. Se determinó a su vez la velocidad de propagación de la onda $c_0 = 2\sqrt{D_I\beta(S_0 - \gamma/\beta)}$ y se estableció la jerarquía $S_0 > \gamma/\beta > S_1 > 0$ para la existencia de la onda.

2.3.2. Soluciones en sistema de reacción-difusión-convección

En este trabajo exploramos algunos resultados sobre un modelo SIR como el presentado en la sección 2.3 pero con un término adicional de convección. Esto es,

$$\partial_t S = -\beta SI + D_S\nabla^2 S \tag{2.15}$$

$$\partial_t I = \beta SI + D_I\nabla^2 I - \mathbf{v}\cdot\nabla I \tag{2.16}$$

donde $\mathbf{v}$ es un campo vectorial. En términos epidemiológicos no es fácil dar una interpretación a este campo vectorial, sin embargo, podría interpretarse como un campo de migración de individuos infectados de un lugar a otro y la dirección de $\mathbf{v}$ sería la dirección de la migración, en tanto que su magnitud sería la velocidad de migración. Pensando en incendios es más sencillo motivar este término de convección, pues se podría interpretar como el campo de viento.

Ahora bien, se puede ver que este modelo también tiene soluciones de onda cuando $\mathbf{v}$ es constante. Sin perder generalidad podemos tomar $\mathbf{v} = (v, 0)$ con $v > 0$, de modo que la ecuación de evolución para I queda

$$\partial_t I = \beta SI + D_I\nabla^2 I - v\partial_x I. \tag{2.17}$$

Pasando las ecuaciones a un sistema de coordenadas que se mueve con la velocidad v, es decir, $x' = x - vt$, se obtiene

$$\partial_t I = \beta SI + D_I\nabla^2 I, \tag{2.18}$$

es decir, recuperamos el sistema de reacción-difusión que estudiamos antes. Tomando el desarrollo de la sección anterior se obtiene en este caso que la velocidad de propagación del frente de infección es

$$c = c_0 + v,$$

mientras que si tomamos viento en contra $\mathbf{v} = (-v, 0)$ se tiene,

$$c = c_0 - v.$$

Lo interesante del caso con viento en contra es que debemos pedir que $c > 0$ de lo cual surge un nuevo valor crítico para el umbral de propagación. Es decir,

$$\begin{aligned} c &> 0 \\ 2\sqrt{D_I\beta(S_0 - \gamma/\beta)} - v &> 0 \\ \sqrt{D_I\beta(S_0 - \gamma/\beta)} &> v/2 \\ \beta &> \frac{1}{S_0}(\frac{v^2}{4D_I} + \gamma) \equiv \beta_c, \end{aligned}$$

donde β_c es el umbral de propagación para el caso con viento en contra.

2.4. Modelo SIR de reacción-difusión heterogéneo

En esta sección presentamos el modelo que estudiamos con profundidad en este trabajo junto a algunas descripciones estadísticas del mismo que se usarán posteriormente.

Esencialmente las ecuaciones son las mismas que 2.6 y 2.7 con la salvedad de que ahora introducimos hetereogeneidad en el medio poniendo una tasa de transmisión $\beta_{\mathbf{r}}$ dependiente de la posición. Por completitud escribimos las ecuaciones nuevamente aquí,

$$\frac{\partial S}{\partial t} = -\beta_{\mathbf{r}} SI + D_S \nabla^2 S, \tag{2.19}$$

$$\frac{\partial I}{\partial t} = \beta_{\mathbf{r}} SI - \gamma I + D_I \nabla^2 I. \tag{2.20}$$

Una tasa de transmisión $\beta_{\mathbf{r}}$ de este tipo podría utilizarse, por ejemplo, para estudiar el efecto de la vacunación sobre la población. Esto es entendible dado que se espera que los lugares donde hay población vacunada la tasa de transmisión sea menor ya que hay menos individuos susceptibles para infectarse. Otra alternativa apuntaría a describir lugares donde la tasa de transmisión es más baja por un efecto de densidad poblacional. En cuanto a la propagación de incendios podría interpretarse de manera similar como una variación de la densidad de vegetación en el espacio, lo cual facilitaría o no la transmisión de las llamas.

2.4.1. Problema y observables

Vamos a centrarnos en estudiar el frente de infección/incendio en un espacio cuadrado de longitud L, con $x, y \in [0, L-1]$, y condiciones iniciales

$$I(x, y, 0) = I_0, \qquad S(x, y, 0) = 1 - I_0,$$

con $x \in (0, \delta x)$ y

$$I(x, y, 0) = 0, \qquad S(x, y, 0) = S_0,$$

para $x \in [\delta x, L-1)$. Adicionalmente, se tienen condiciones de contorno de Dirichlet en la dirección x,

$$I(0, y, t) = I(L-1, y, t) = S(0, y, t) = S(L-1, y, t) = 0.$$

Y condiciones periódicas en la dirección y,

$$I(x,0,t) = I(x,L-1,t) \qquad\qquad S(x,0,t) = S(x,L-1,t).$$

Estas condiciones iniciales y de contorno dan lugar a un único frente de onda propagándose en la dirección x, lo cual resulta conveniente para realizar un análisis estadístico a partir de ciertos observables, los cuales se definen a continuación.

Para caracterizar las fluctuaciones espaciales y temporales del frente de infección/incendio, definimos el campo de desplazamiento del frente $u(y,t)$ como

$$\max_{x\in(0,L-1)}\{I(x,y,t)\} = I(u(y,t),y,t). \tag{2.21}$$

Es decir, $u(y,t)$ es la posición en x del máximo de la fracción de infectados/incendiados para un dado y en el instante t. Se define a su vez el centro de masa de $u(y,t)$,

$$u_{cm}(t) \equiv \langle u(y,t)\rangle_y, \tag{2.22}$$

donde $\langle ...\rangle_y$ indica el promedio sobre la coordenada y. El valor medio de la amplitud máxima de $I(x,y,t)$ sobre $u(y,t)$ es,

$$I_{max}(t) = \langle I(u(y,t),y,t)\rangle_y. \tag{2.23}$$

Por otro lado, la velocidad media de la onda de propagación se define como,

$$c \equiv \langle \dot{u}_{cm}(t)\rangle_t. \tag{2.24}$$

donde $\langle ...\rangle_t$ indica promedio sobre el tiempo. De manera similar, la amplitud media del frente de propagación es

$$I_{max} = \langle I_{max}(t)\rangle_t. \tag{2.25}$$

Las fluctuaciones del frente de onda pueden ser caracterizadas definiendo la rugosidad del mismo como la desviación estándar de $u(y,t)$,

$$w(t)^2 \equiv \langle [u(y,t) - u_{cm}(t)]^2\rangle_y, \tag{2.26}$$

o con el factor de estructura,

$$S(q) \equiv \langle |u(q,t)|^2\rangle_t, \tag{2.27}$$

donde $u(q,t)$ es la transformada de Fourier sobre el espacio de $u(y,t)$. Por último, estaremos interesados en observar el perfil de la onda de propagación, estos es,

$$f_I(x) = \langle I(x - u(y,t),y,t)\rangle_{y,t}. \tag{2.28}$$

En el capítulo **??** utilizaremos los observables definidos aquí como principales herramientas para caracterizar y comparar de manera cuantitativa los efectos que tienen distintas heterogeneidades, introducidas mediante $\beta_{\mathbf{r}}$, sobre la dinámica del problema.

Capítulo 3

Reseña numérica y computacional

La idea de este capítulo es dejar documentación que muestre con claridad el trabajo de investigación y elaboración técnico, a nivel computacional, realizado para llevar a cabo este proyecto. Adicionalmente, está en mi intención ser lo más genérico, claro y acotado posible, para que sea de utilidad a cualquier otra persona que esté interesada en resolver ecuaciones diferenciales parciales haciendo uso de programación en paralelo. En particular, usando *CUDA* [20] a través de las bondades que ofrece *Python* [21] mediante la librería *CuPy* [22].

Este capítulo cuenta con un material complementario en formato de *Jupyter Notebook* en *Google Colab* al cual puede acceder desde aquí. *Google Colab* da acceso gratuito a GPUs, lo cual está fantástico para aprender a usarlas, aunque evidentemente es con tiempo limitado. En este *Jupyter Notebook* esencialmente encontrará todo el código presentado aquí y un poco más, para que pueda interactuar y hacer las modificaciones que quiera.

A continuación dejamos constancia del *software* y *hardware* utilizado en la ejecución de los códigos de este capítulo.

- Python: 3.9.7
- CUDA Version: 11.6
- CPU: AMD Ryzen 9 5900HX
- GPU: NVIDIA GeForce RTX 3060 Laptop
- Memoria de GPU: 6GB
- CUDA cores: 3840
- RAM: 32GB

3.1. Diferencias finitas

El objetivo es resolver numéricamente ecuaciones diferenciales parciales de la manera más simple y eficiente posible. Para la física este tipo de ecuaciones son de gran interés, ya que se usan ampliamente para modelar todo tipo de fenómenos.

Para reducir la complejidad del problema, acotamos la discusión mostrando en detalle el proceso de resolución de un sistema de dos ecuaciones de reacción-difusión con dos dimensiones espaciales (x, y) en cierta región $\Omega \in \mathbb{R}^2$. Esto es conveniente porque este problema corresponde con el tipo de sistemas usados en este trabajo. Explícitamente, queremos resolver,

$$\begin{aligned} \partial_t u &= f_u(u, v) + D_u \nabla^2 u \\ \partial_t v &= f_v(u, v) + D_v \nabla^2 v, \end{aligned} \tag{3.1}$$

donde u y v son las variables dinámicas que dependen de las variables (t, x, y), f_u y f_v son funciones suaves, mientras que D_u y D_v son los coeficientes de difusión de u y v respectivamente. Estas ecuaciones deben resolverse teniendo en cuenta determinadas condiciones iniciales y de contorno para el problema en cuestión, podemos expresarlas genéricamente de la siguiente manera,

$$\begin{array}{llll} u(t=0,x,y)=g_u(x,y) & \forall x,y\in\Omega & | & u(t,x,y)=h_u(t,x,y) \quad \forall t\in\mathbb{R};\ \forall x,y\in\delta\Omega \\ v(t=0,x,y)=g_v(x,y) & \forall x,y\in\Omega & | & v(t,x,y)=h_v(t,x,y) \quad \forall t\in\mathbb{R};\ \forall x,y\in\delta\Omega. \end{array}$$

Donde las funciones g_u y g_v denotan las condiciones iniciales del sistema, las funciones h_u y h_v las condiciones de contorno y $\delta\Omega$ es el borde de Ω.

De esta manera queda completamente definido el problema y procedemos al armado del esquema numérico necesario para resolverlo. Por simplicidad consideramos que Ω es una región rectangular del plano donde $x, y \in [0, L_x]\times[0, L_y]$. Segmentamos este espacio en $N_x \times N_y$ cuadrantes de dimensiones $d_x = L_x/N_x$ y $d_y = L_y/N_y$ y denominamos u_{ij} y v_{ij} a los valores de las funciones u y v a tiempo t en el cuadrante (i, j) correspondiente a la región $[jd_x, (j+1)d_x) \times [id_y, (i+1)d_y)$, con $i = 0, 1, ..., N_y - 1$ y $j = 0, 1, ..., N_x - 1$.

A continuación, la idea consiste en reducir el sistema de ecuaciones diferenciales parciales 3.1 en un sistema de ecuaciones diferenciales ordinarias, donde cada ecuación describe la evolución de las variables dinámicas en un cuadrante distinto. Para ello necesitamos llevar los laplacianos de las ecuaciones al nuevo esquema espacial discretizado. Lo hacemos usando la siguiente aproximación por diferencias finitas para los laplacianos,

$$\begin{aligned} (\nabla^2 u)_{ij} &= \frac{1}{d^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}) \\ (\nabla^2 v)_{ij} &= \frac{1}{d^2}(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{ij}) \end{aligned} \tag{3.2}$$

donde usamos que $d = d_x = d_y$. Utilizando la notación dada para la discretización espacial, tenemos el siguiente sistema de ecuaciones diferenciales ordinarias,

$$\begin{aligned} \frac{\mathrm{d}u_{ij}}{\mathrm{d}t} &= f_u(u_{ij}, v_{ij}) + D_u(\nabla^2 u)_{ij} \\ \frac{\mathrm{d}v_{ij}}{\mathrm{d}t} &= f_v(u_{ij}, v_{ij}) + D_v(\nabla^2 v)_{ij}. \end{aligned} \tag{3.3}$$

Finalmente, discretizamos el espacio temporal con un intervalo dt y aproximamos la derivada temporal a primer orden. Usando $n \in \mathbb{N}$ como índice temporal, notamos u^n_{ij} y v^n_{ij} como el valor de las funciones u y v en el instante $t = n{*}dt$ sobre el cuadrante (i, j). De esta manera obtenemos el siguiente esquema explícito de Euler para la resolución numérica del sistema 3.1,

$$\begin{aligned} u^{n+1}_{ij} &= u^n_{ij} + dt\ \left(f_u(u^n_{ij}, v^n_{ij}) + D_u(\nabla^2 u)^n_{ij}\right) \\ v^{n+1}_{ij} &= v^n_{ij} + dt\ \left(f_v(u^n_{ij}, v^n_{ij}) + D_v(\nabla^2 v)^n_{ij}\right), \end{aligned} \tag{3.4}$$

a partir del cual podemos iterar para obtener la solución. En la siguiente sección se describe cómo hacer esto usando *Python* con diferentes niveles de eficiencia.

3.2. Implementaciones en *Python*

En esta sección se describen brevemente 4 implementaciones distintas para la resolución del esquema numérico 3.4 hallado en la sección anterior. Comenzamos con una implementación usando la librería *NumPy* [23] en un esquema

serial, luego vamos optimizando las implementaciones hasta llegar a la más eficiente obtenida, correspondiente a la utilización de *CUDA* a través de la librería *CuPy*.

La razón para hacer esto reside nuevamente en la idea de dejar documentación que pueda llegar a ser de utilidad para alguien más lidiando con problemas numéricos donde la eficiencia de la resolución es relevante. Además, antes de llegar a la versión final, donde usamos *CUDA* y consecuentemente es necesario un procesador gráfico (GPU) compatible con *CUDA*, se muestran implementaciones que son notablemente eficientes sin necesidad de una GPU, y que pueden ser usadas por una computadora más modesta.

3.2.1. Implementación con *NumPy*

NumPy es una de las librerías fundamentales para hacer computación científica en *Python*. Ofrece los invaluables *NumPy Arrays*, que están diseñados específicamente para ser lo más eficiente posible en la manipulación numérica sin perder la simplicidad y elegancia de *Python*. Utilizando *NumPy* se obtienen mejores resultados que usando *Python* nativo porque integra código de *C,C++ y Fortran* dentro de *Python*, adicionalmente la mayoría de los métodos implementados para la operación con *NumPy arrays* están paralelizados.

El código 3.1 muestra la implementación propuesta usando *NumPy*. Nótese que hay más líneas de comentarios que de código. Esencialmente, primero definimos una función que llamamos *laplacian*, que toma un *array* 2D y devuelve su laplaciano, para ello usamos la función *roll* de *NumPy* que desplaza un *array* sobre un eje dado cuanto queramos y con condiciones periódicas. Luego, sumamos y restamos estas matrices desplazadas según 3.2 y obtenemos el laplaciano. Finalmente, definimos la función *cpu_ numpy_ solver*, que toma por argumentos las condiciones iniciales del problema, las funciones de reacción f_u y f_v, los coeficientes de difusión, el intervalo de integración temporal dt y espacial d y la cantidad de iteraciones it.

```python
import numpy as np

def laplacian(X):
  '''
  Take the Laplacian of a 2D array with periodic boundary conditions.
  '''
  return np.roll(X,1,axis = 0) + np.roll(X,-1,axis = 0) + np.roll(X,1,axis = 1) + np.roll(X,-1,axis = 1) - 4*X

def cpu_numpy_solver(u, v, fu, fv, Ds, dt = .01, d = 1, it = 1000):
  '''
  Solve a 2D reaction-diffusion equation for two dynamical variables with periodic boundary conditions using NumPy.

  u: Intial conditions for the first dynamical variable. 2d NumPy array.
  v: Intial conditions for the second dynamical variable. 2d NumPy array.
  fu: Reaction term for the first dynamical variable. Function.
  fv: Reaction term for the second dynamical variable. Function.
  Ds: Diffusion coefficients for the first and second dynamical variables. List or array of length 2.
  dt: Time step. Default value is 0.01.
  d: Space step. Default value is 1.
  it: Number of iterations. Default value is 1000.
  '''

  Du,Dv = Ds
  for _ in range(it):
      u = u + dt*(fu(u,v) + Du*laplacian(u)/d**2)
      v = v + dt*(fv(u,v) + Dv*laplacian(v)/d**2)

  return u,v
```

Código 3.1: Implementación con NumPy.

```python
def fu(u,v,beta):
  return -beta*u*v

def fv(u,v,beta,gamma):
  return beta*u*v - gamma*v

L = 1024
beta = 1
```

```
gamma = .2
u = np.ones((L,L))
v = np.zeros((L,L))
u[:,0] = 0
v[:,0] = 1
Ds = [0,1]

uf,vf = cpu_numpy_solver(u,v,fu,fv,Ds)
```

Código 3.2: Ejemplo de uso de la implementación con *NumPy*.

En el código 3.2 se muestra un ejemplo de uso utilizando las siguientes funciones de reacción,

$$\begin{aligned} f_u(u,v) &= -\beta uv, \\ f_v(u,v) &= \beta uv - \gamma v, \end{aligned} \tag{3.5}$$

donde β y γ son constantes.

Finalmente, a continuación se muestra la velocidad de la resolución con un sistema de 1024×1024 utilizando todos los parámetros y funciones del ejemplo 3.2. Para una justa comparación con las demás implementaciones, utilizaremos exactamente los mismos parámetros y funciones.

```
  %timeit cpu_numpy_solver(u,v,fu,fv,Ds)
  1min +- 1.53 s per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

3.2.2. Implementación serial con *Numba*

Otra de las librerías fundamentales para cálculo numérico en *Python* es *Numba* [24]. *Numba* toma código nativo de *Python* y genera código de máquina optimizado usando *LLVM compiler infrastructure* [25]. También es capaz de procesar *NumPy* con una gran cantidad de sus métodos.[1]

Lo mejor de todo esto es que *Numba* lo hace de manera completamente autónoma, solo hay que decirle que lo haga. Para ello utilizamos el decorador *@njit*, que indica a *Numba* que la función en cuestión debe ser procesada. El código 3.3 muestra cómo sería la implementación en este caso. Se define la función *cpu_numba_solver* que toma los mismos argumentos que la función *cpu_numpy_solver* vista anteriormente. Dentro de la función iteramos temporalmente y para calcular los laplacianos y términos de reacción en cada cuadrante recorremos las matrices con un ciclo *for*. Usualmente es posible tomar las funciones o implementaciones realizadas con *Numpy* y decorarlas con *@njit* para obtener el resultado deseado. Sin embargo, en este caso no podemos hacerlo con las funciones del código 3.1 porque *Numba* no soporta la función *roll* de *NumPy*. Por lo cual estamos forzados a calcular el laplaciano de una manera más explicita para que *Numba* lo entienda.

```
from numba import njit
import numpy as np

@njit()
def cpu_numba_solver(u, v, fu, fv, Ds, dt=.01, d = 1, it = 1000):
  '''
  Solve a 2D reaction-diffusion equation for two dynamical variables with periodic boundary conditions using Numba
  in a serial implementation.

  u: Intial conditions for the first dynamical variable. 2d NumPy array of shape (Ly,Lx).
  v: Intial conditions for the second dynamical variable. 2d NumPy array of shape (Ly,Lx).
  fu: Reaction term for the first dynamical variable. Function.
  fv: Reaction term for the second dynamical variable. Function.
  Ds: Diffusion coefficients for the first and second dynamical variables. List or array of length 2.
  dt: Time step. Default value is 0.01.
  d: Space step. Default value is 1.
  it: Number of iterations. Default value is 1000.
  '''

  Ly,Lx = u.shape
```

[1] Para ver más detalles consultar la documentación de *Numba*.

```
  u = u.reshape(Ly*Lx)
  v = v.reshape(Ly*Lx)
  Fu = np.zeros_like(u)
  Fv = np.zeros_like(v)
  Du,Dv = Ds
  for _ in range(it):
    for i in range(Lx*Ly):
      x = i % Lx
      y = i // Lx
      Lu = (u[(x+1)%Lx + Lx*y] + u[(x-1+Lx)%Lx+Lx*y] + u[x + Lx*((y+1)%Ly)] + u[x + Lx*((y-1+Ly)%Ly)] - 4*u[i])/d**2
      Lv = (v[(x+1)%Lx + Lx*y] + v[(x-1+Lx)%Lx+Lx*y] + v[x + Lx*((y+1)%Ly)] + v[x + Lx*((y-1+Ly)%Ly)] - 4*v[i])/d**2
      Fu[i] = fu(u[i],v[i]) + Du*Lu
      Fv[i] = fv(u[i],v[i]) + Dv*Lv
    u = u + dt*Fu
    v = v + dt*Fv
  return u.reshape(Ly,Lx),v.reshape(Ly,Lx)
```

Código 3.3: Implementación serial con *Numba*.

En cuanto al ejemplo de uso, sería idéntico al mostrado en 3.2, con la única salvedad de que las funciones f_u y f_v también deben estar decoradas con *@njit*. A continuación se muestra el tiempo de resolución para sistemas de 1024×1024 en esta versión. Resulta cerca de 3 veces más rápido que 3.1.

```
%timeit cpu_numba_solver(u,v,fu,fv,Ds)
18.1 s +- 564 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

3.2.3. Implementación paralela con *Numba*

En esta ocasión la idea es utilizar *Numba* aprovechando que una parte del algoritmo se puede realizar en paralelo. Esto simplemente quiere decir que hay un conjunto de operaciones que puede realizarse en simultáneo en lugar de una por una. Este es el caso para el ciclo *for* que se utiliza para calcular los laplacianos y términos de reacción en cada cuadrante. Es decir, no es necesario calcular el valor correspondiente al cuadrante (i, j) para luego calcular el del cuadrante (i', j'), se pueden calcular simultáneamente, en paralelo.

Lo mejor del caso, nuevamente, es que podemos indicarle de una manera muy sencilla a *Numba* que determinado *for* es paralelizable y listo, *Numba* se encargará de darle las instrucciones en paralelo al procesador. Para ello, todo lo que tenemos que hacer es importar la función *prange* de *Numba* que sirve para indicarle a *Numba* que el ciclo *for* es paralelizable y pasar la opción *parallel* = *True* al decorador *@njit*. Por completitud mostramos el código de la función *cpu_numba_parallel_solver* en 3.4, pero las únicas diferencias con la versión serial 3.3 son las indicadas aquí.

```
from numba import njit,prange
import numpy as np

@njit(parallel = True)
def cpu_numba_parallel_solver(u, v, fu, fv, Ds, dt=.01, d = 1, it = 1000):
  '''
  Solve a 2D reaction-diffusion equation for two dynamical variables with periodic boundary conditions using Numba
  with a parallel implementation.

  u: Intial conditions for the first dynamical variable. 2d NumPy array of shape (Ly,Lx).
  v: Intial conditions for the second dynamical variable. 2d NumPy array of shape (Ly,Lx).
  fu: Reaction term for the first dynamical variable. Function.
  fv: Reaction term for the second dynamical variable. Function.
  Ds: Diffusion coefficients for the first and second dynamical variables. List or array of length 2.
  dt: Time step. Default value is 0.01.
  d: Space step. Default value is 1.
  it: Number of iterations. Default value is 1000.
  '''

  Ly,Lx = u.shape
  u = u.reshape(Ly*Lx)
  v = v.reshape(Ly*Lx)
  Fu = np.zeros_like(u)
  Fv = np.zeros_like(v)
  Du,Dv = Ds
```

```
26
27  for _ in range(it):
28    for i in prange(Lx*Ly):
29      x = i % Lx
30      y = i // Lx
31      L_u = (u[(x+1)%Lx + Lx*y] + u[(x-1+Lx)%Lx+Lx*y] + u[x + Lx*((y+1)%Ly)] + u[x + Lx*((y-1+Ly)%Ly)] - 4*u[i])/d**2
32      L_v = (v[(x+1)%Lx + Lx*y] + v[(x-1+Lx)%Lx+Lx*y] + v[x + Lx*((y+1)%Ly)] + v[x + Lx*((y-1+Ly)%Ly)] - 4*v[i])/d**2
33      Fu[i] = fu(u[i],v[i]) + Du*L_u
34      Fv[i] = fv(u[i],v[i]) + Dv*L_v
35    u = u + dt*Fu
36    v = v + dt*Fv
37  return u.reshape(Ly,Lx),v.reshape(Ly,Lx)
```

Código 3.4: Implementación paralela con *Numba*.

A continuación mostramos el rendimiento obtenido con esta nueva versión, observamos que es cerca de 3 veces más rápido que la versión serial 3.3 y 10 veces más rápido que la versión de *NumPy* 3.1. Esto muestra una primera aproximación al poder de la programación en paralelo y cómo es posible implementarla sin la necesidad de una GPU.

```
%timeit cpu_numba_parallel_solver(u,v,fu,fv,Ds)
6.31 s +- 439 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

3.2.4. Implementación con *CuPy*

CuPy es una librería de código abierto desarrollada para facilitar el acceso a las GPU compatibles con *CUDA*[2] en *Python*. *Cupy* ofrece prácticamente todos los métodos de *NumPy* y se encarga de lidiar de forma autónoma y eficiente con el problema de pasar las instrucciones a la GPU. Esta característica por sí sola ya es sorprendente, dado que ofrece la posibilidad de explotar las capacidades de la GPU sin saber nada de CUDA. Ello quiere decir, que en muchos casos basta escrbir el código tal como lo haríamos con *NumPy* pero usando *CuPy*, y eso bastaría para tener una mejora decisiva en la eficiencia. De hecho, hagamos la prueba, si tomamos el código 3.1 y lo único que hacemos es cambiar *NumPy* por *CuPy*, el resultado obtenido es el siguiente.

```
%timeit gpu_simple_cupy_solver(u,v,fu,fv,Ds)
3.12 s +- 3.51 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

Esto es cerca de 19 veces más rápido que la versión de *NumPy* y además la más rápida hasta ahora, lograda con un esfuerzo mínimo. Cabe recordar que el tamaño del sistema que estamos resolviendo, es de 1024×1024 en todos los casos, este es un terreno favorable para el uso de GPU, dado que es un sistema lo suficientemente grande como para que la capacidad de paralelización masiva de la GPU marque la diferencia. Esto es de carácter elemental en lo que respecta al uso de procesadores gráficos para cálculo numérico, sin embargo no está de más recordarlo. No siempre es preferible usar la GPU, hay que ponderar el carácter del algoritmo a utilizar y la magnitud de operaciones susceptibles de ser paralelizadas. Si corremos los mismos códigos con un sistema de 100×100, las cosas cambian rotundamente.

```
#Sistemas de 100x100
#Numpy
%timeit cpu_numpy_solver(u,v,fu,fv,Ds)
163 ms +- 502 us per loop (mean +- std. dev. of 7 runs, 10 loops each)
#Cupy
%timeit gpu_simple_cupy_solver(u,v,fu,fv,Ds)
1.07 s +- 3.91 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

Ahora bien, volviendo a sistemas grandes, podría decirse que una mejora en un factor 19 es sorprendente, sin embargo esto es así solo si comparamos con la implementación de *NumPy*, pero si miramos la mejora respecto de la implementación con *Numba* en paralelo, el factor de mejora es cerca de 2. En este caso alguien podría decir, con razón, que la utilización de la *GPU* no está justificada, dado que la mejora en eficiencia no vale el costo que implica el acceso a la GPU.

Para sortear esta razonable objeción, solo debemos profundizar un poco más en las herramientas que nos ofrece *CuPy*. Como decíamos al principio, el hecho de que *CuPy* ofrezca la posibilidad de acceder a la computación en GPU

[2]Está en fase experimental la posibilidad de usar GPUs con ROCm.

de una manera extremadamente sencilla y con buenos resultados, es sorprendente. Sin embargo, como es habitual, esa sencillez no viene gratis, paga un peaje a la potencia de cómputo real que puede ofrecer una GPU.

En lo que sigue se muestra cómo es posible sacar más provecho de la GPU, sin salirnos del entorno que ofrece *CuPy*. La razón por la que estamos desperdiciando potencia de cómputo al resolver el problema reemplazando *CuPy* por *NumPy* es porque estamos lanzando demasiados *kernels* de manera de innecesaria. Es radicalmente más eficiente si conseguimos juntar todas las operaciones necesarias en una menor cantidad de *kernels*. Por cierto, se le dice *kernel* a una función que se ejecuta en la GPU, comúnmente esta función representan operaciones elementales que se realizan en paralelo en la GPU. Para entender esto, vamos a ver un ejemplo sencillo antes de atacar el problema original.

Supongamos que queremos multiplicar las matrices A y B, elemento a elemento, y luego sumar el resultado a la matriz C. Utilizando *CuPy arrays* la función en cuestión y el resultado obtenido es el siguiente.

```
import cupy as cp
def mul_add(A,B,C):
    return A*B + C

L = 1024
A = cp.ones((L,L))
B = cp.ones((L,L))
C = cp.ones((L,L))
mul_add(A,B,C)
%timeit mul_add(A,B,C)
215 us +- 6.81 us per loop (mean +- std. dev. of 7 runs, 1,000 loops each)
```

Ahora bien, podemos hacerlo mejor, el problema con la función anterior es que está utilizando dos *kernels* en lugar de uno, uno para multiplicar y el otro para sumar. Sin embargo sería mejor si pudiéramos usar un solo *kernel* que hiciera las dos cosas a la vez. Para ello, *CuPy* ofrece la posibilidad de escribir *kernels* personalizados, una de las maneras de hacerlo, es utilizando la función *cupy.ElementwiseKernel*. A continuación se muestra cómo quedaría la función y su resultado utilizando esta alternativa.

```
import cupy as cp
mul_add_kernel = cp.ElementwiseKernel(
  'float64 A, float64 B, float64 C', 'float64 out',
  'out = A*B + C', 'mul_add')

L = 1024
A = cp.ones((L,L))
B = cp.ones((L,L))
C = cp.ones((L,L))
mul_add_kernel(A,B,C)
%timeit mul_add_kernel(A,B,C)
136 us +- 273 ns per loop (mean +- std. dev. of 7 runs, 10,000 loops each)
```

Vemos que la velocidad aumentó apreciablemente aún en un ejemplo tan sencillo como este, si aplicamos esta misma idea a algoritmos más complejos tenemos la posibilidad de mejorar la eficiencia sorprendentemente. No voy a entrar en los detalles de utilización de la función *cupy.ElementwiseKernel*, para más detalles recomiendo la siguiente documentación.

Ahora procedemos finalmente con la propuesta para la resolución del problema original usando esta nueva idea de fusionar *kernels* (Código 3.5). Para ello lo que haremos será concentrar en un único *kernel* los cálculos necesarios para evaluar las funciones de reacción y los laplacianos. Nuevamente, no me detendré a explicar los detalles de la implementación, pero espero que el código sea lo suficientemente claro como para motivar el interés del lector.

```
import cupy as cp

forces = cp.ElementwiseKernel(
  'raw float64 u, raw float64 v, float64 beta, float64 gamma, float64 Du, float64 Dv, uint32 Lx, uint32 Ly',
  'float64 Fu, float64 Fv',
  '''
  int x = i % Lx;
  int y = (int) i / Lx;
  Fu = -beta*u[i]*v[i] + Du*(u[(x+1)%Lx+Lx*y] + u[(x-1+Lx)%Lx+Lx*y] + u[x+Lx*((y+1)%Ly)] + u[x+Lx*((y-1+Ly)%Ly)]
        - 4*u[i]);
  Fv = beta*u[i]*v[i] - gamma*v[i] + Dv*(v[(x+1)%Lx + Lx*y] + v[(x-1+Lx)%Lx+Lx*y] + v[x + Lx*((y+1)%Ly)]
        + v[x + Lx*((y-1+Ly)%Ly)] - 4*v[i]);
```

```
  ''',
  'forces')

euler = cp.ElementwiseKernel(
  'float64 Fu, float64 Fv, float64 dt','float64 u, float64 v',
  '''
  u = u + dt*Fu;
  v = v + dt*Fv;
  ''',
  'euler')

def gpu_cupy_solver(u, v, Ds, beta = 1.,gamma = .2, dt = .01, d = 1, it = 1000):
  '''
  Solve a 2D reaction-diffusion equation for two dynamical variables with periodic boundary conditions using CuPy.

  u: Intial conditions for the first dynamical variable. 2d CuPy array of shape (Ly,Lx).
  v: Intial conditions for the second dynamical variable. 2d CuPy array of shape (Ly,Lx).
  Ds: Diffusion coefficients for the first and second dynamical variables. List or array of length 2.
  dt: Time step. Default value is 0.01.
  d: Space step. Default value is 1.
  it: Number of iterations. Default value is 1000.
  '''
  Ly,Lx = u.shape
  Du,Dv = Ds
  Fu = cp.zeros_like(u)
  Fv = cp.zeros_like(v)

  for _ in range(it):
    forces(u,v,beta,gamma,Du,Dv,Lx,Ly,Fu,Fv)
    euler(Fu,Fv,dt,u,v)
  return u,v
```

Código 3.5: Implementación con *CuPy*.

Este es el tipo de implementación utilizada en este proyecto de tesis, el resultado obtenido en esta ocasión es el siguiente,

```
  %timeit gpu_cupy_solver(u,v,Ds)
  511 ms +- 12.3 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

esto es, mejor en un factor 6 que la versión estándar de *CuPy*, 12 veces mejor que la implementación con *Numba* en paralelo (la mejor sin usar GPU), y 117 veces más rápido que la versión con *NumPy*. Ahora sí, vemos que la diferencia entre usar una GPU y no usarla es, por lo menos, en un factor de 12, la diferencia entre una simulación de un 1 día y una de 12 días.

3.3. Tamaño del sistema

Por último, quisiera terminar este capítulo con algunos comentarios. Por un lado, recordar que todas las comparaciones de velocidad en las diferentes implementaciones se realizaron con los mismos parámetros, y fundamentalmente con sistemas relativamente grandes, 1024×1024, donde la GPU se ve favorecida sobre la CPU. Diferente es el caso para sistemas chicos, por completitud en este sentido, en la figura **??** mostramos los resultados de velocidad de las distintas implementaciones para distintos tamaños de sistemas. Por otro lado, la mayoría de los resultados presentados en este proyecto fueron obtenidos con sistemas de $2^{16} \times 2^{11}$, en un sistema de dos ecuaciones de reacción-difusión como el discutido aquí, esto implica cuatro matrices con 2^{27} elementos cada una, considerando que además utilizamos un formato en coma flotante de doble precisión, cada matriz ocupa alrededor de $1GB$ de memoria. El tiempo que llevaría resolver sistemas de este tamaño sin GPU sería completamente inviable.

Finalmente, es posible que quienes estén más interiorizados con *CUDA* tal vez estén sorprendidos por el carácter superficial con el que usamos la GPU aquí. Es decir, cuando escribimos el *kernel* no lo escribimos en *CUDA C* o *CUDA C++*, es algo similar, pero es distinto, lo cual puede confundir un poco. Además, en ningún momento indicamos cantidad de hilos por bloque ni cantidad de bloques en la grilla. Esto es, nuevamente, porque *CuPy* intenta simplificar todo lo posible el uso de la GPU, para que sea accesible a usuarios sin conocimientos de *CUDA*.

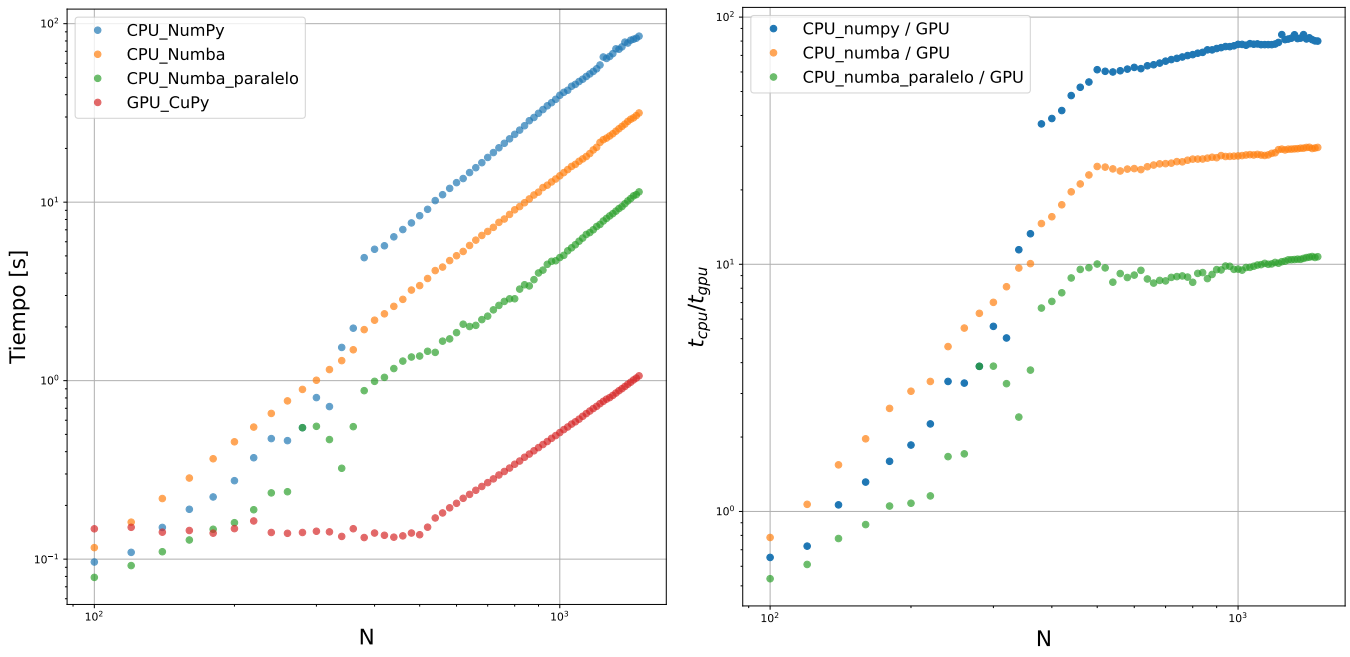

Figura 3.1: **(a)** Tiempo de simulación en función del tamaño del sistema $N \times N$ para las distintas implementaciones. **(b)** Aceleración obtenida al usar GPU en función del tamaño del sistema $N \times N$ para las distintas implementaciones.

Sin embargo, *CuPy* ofrece también todo el acceso al detalle, tal como podría hacerse con *CUDA C* por ejemplo, y admite la escritura de *kernels* directamente en *CUDA C* a través de la función *cupy.RawKernel*. En el material complementario de este capítulo, dejo la implementación usando esta función que da acceso completo a *CUDA*. No la agrego acá porque entiendo que no aporta demasiado, adicionalmente, no pude obtener ninguna mejora notable respecto al *kernel* escrito con *cupy.ElementwiseKernel*.

Bibliografía

[1] Bressan, A., De Lellis, C. Existence of optimal strategies for a fire confinement problem. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, **62** (6), 789–830, 2009. 1

[2] Riley, S., Eames, K., Isham, V., Mollison, D., Trapman, P. Five challenges for spatial epidemic models. *Epidemics*, **10**, 68–71, 2015. Challenges in Modelling Infectious Disease Dynamics. 1

[3] Kermack, W. O., McKendrick, A. G., Walker, G. T. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, **115** (772), 700–721, 1927. 1, 4

[4] Keeling, M. J., Rohani, P. Modeling Infectious Diseases in Humans and Animals. págs. 15–26. Princeton University Press, 2008. 5, 7

[5] Noble, J. V. Geographic and temporal development of plagues. *Nature*, **250**, 726–729, 1974.

[6] Kolton, A. B., Laneri, K. Rough infection fronts in a random medium. *The European Physical Journal B*, **92** (6), Jun 2019. 1

[7] Abramson, G., Kenkre, V., Yates, T., Parmenter, R. Traveling waves of infection in the hantavirus epidemics. *Bulletin of mathematical biology*, **65**, 519–34, 06 2003. 1

[8] Romeo-Aznar, V., Paul, R., Telle, O., Pascual, M. Mosquito-borne transmission in urban landscapes: The missing link between vector abundance and human density. *Proceedings of the Royal Society B: Biological Sciences*, **285**, 20180826, 08 2018. 1

[9] Cesana, D., Benedictow, O., Bianucci, R. The origin and early spread of Black Death in Italy: first evidence of plague victims from 14th century Liguria (Northern Italy). *Anthropological Science (Japanese Series)*, **125**, 03 2017. 4

[10] Christakos, G., Olea, R. A., Serre, M. L., Wang, L.-L., Yu, H.-L. Interdisciplinary public health reasoning and epidemic modelling: the case of black death. Springer, 2005. 3

[11] Murray, J. D. Mathematical Biology II: Spatial Models and Biomedical Applications, tomo 18 de *Interdisciplinary Applied Mathematics*. Springer New York, 2003. 3, 7

[12] Taubenberger, J. K. The origin and virulence of the 1918 "Spanish" influenza virus. *Proceedings of the American Philosophical Society*, **150** (1), 86, 2006. 3

[13] Worobey, M., Han, G.-Z., Rambaut, A. Genesis and pathogenesis of the 1918 pandemic H1N1 influenza A virus. *Proceedings of the National Academy of Sciences*, **111** (22), 8107–8112, 2014. 4

[14] Mills, C. E., Robins, J. M., Lipsitch, M. Transmissibility of 1918 pandemic influenza. *Nature*, **432** (7019), 904–906, 2004. 4

[15] Harko, T., Lobo, F. S., Mak, M. Exact analytical solutions of the Susceptible-Infected-Recovered (SIR) epidemic model and of the SIR model with equal death and birth rates. *Applied Mathematics and Computation*, **236**, 184–194, 2014. 5

[16] Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. Numerical Recipes 3rd Edition: The Art of Scientific Computing. págs. 907–910. Cambridge University Press, 2007. 5

[17] Murray, J. D. Mathematical Biology I. An Introduction, tomo 17 de *Interdisciplinary Applied Mathematics*. New York: Springer, 2002. 7

[18] Crank, J. The mathematics of diffusion. Oxford university press, 1979. 7

[19] Turing, A. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, **237**, 37–72, 1952. 7

[20] NVIDIA, Vingelmann, P., Fitzek, F. H. CUDA, release: 10.2.89, 2020. 12

[21] Van Rossum, G., Drake, F. L. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace, 2009. 12

[22] Okuta, R., Unno, Y., Nishino, D., Hido, S., Loomis, C. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. En: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS). 2017. 12

[23] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., *et al.* Array programming with NumPy. *Nature*, **585** (7825), 357–362, sep. 2020. 13

[24] Lam, S. K., Pitrou, A., Seibert, S. Numba: A llvm-based python jit compiler. En: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, págs. 1–6. 2015. 15

[25] Lattner, C., Adve, V. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. págs. 75–88. San Jose, CA, USA, 2004. 15

Agradecimientos

Agradezco al Instituto Balseiro, a la Universidad Nacional de Cuyo y a la Comisión Nacional de Energía Atómica por hacer posible mis estudios universitarios. Agradezco enteramente a todo el personal de estas instituciones que hacen posible, con gran esfuerzo y a pesar todas las complicaciones dadas por la pandemia, que educación universitaria de altísimo nivel, pública y gratuita llegue a todo el país. Particular agradecimiento para mi director de tesis, Dr. Alejandro Kolton, por su gran apoyo en el desarrollo de este trabajo y buena predisposición para todo. Agradezco también a mis compañeros de camada Amir Zablotsky, Pedro *El Bata* Llauradó, Esteban *El Leches* Acerbo, Marco Madile, Ezequiel Saidman, Martín Famá y Francisco *El Bbto* Gymnich por estar siempre presentes y ayudar a liberar las tensiones de la vida universitaria.