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

11 de Agosto 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

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 de tesis. 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* a través de las bondades que ofrece *Python* mediante la librería *CuPy*.

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 prensentado 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

2.1. 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,

$$\begin{aligned} \partial_t u &= f_u(u, v) + D_u \nabla^2 u \\ \partial_t v &= f_v(u, v) + D_v \nabla^2 v, \end{aligned} \tag{2.1}$$

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 denommina-mos u_{ij} y v_{ij} a los valores de las funciones u y v a tiempo t en el cuadrante (i, j) corespondiente 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 2.1 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,

$$\begin{aligned} (\nabla^2 u)_{ij} &= \frac{1}{d^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}) \\ (\nabla^2 v)_{ij} &= \frac{1}{d^2}(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{ij}) \end{aligned} \tag{2.2}$$

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

$$\begin{aligned} \frac{\mathrm{d}u_{ij}}{\mathrm{d}t} &= f_u(u_{ij}, v_{ij}) + D_u(\nabla^2 u)_{ij} \\ \frac{\mathrm{d}v_{ij}}{\mathrm{d}t} &= f_v(u_{ij}, v_{ij}) + D_v(\nabla^2 v)_{ij}. \end{aligned} \tag{2.3}$$

Finalmente, discretizamos el espacio temporal con un intervalo dt y aproximamos la derivada temporal a primer orden. Usando $n \in \mathbb{N}$ como ínidce 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 2.1,

$$\begin{aligned} u^{n+1}_{ij} &= u^n_{ij} + dt\ \left(f_u(u^n_{ij}, v^n_{ij}) + D_u(\nabla^2 u)^n_{ij}\right) \\ v^{n+1}_{ij} &= v^n_{ij} + dt\ \left(f_v(u^n_{ij}, v^n_{ij}) + D_v(\nabla^2 v)^n_{ij}\right), \end{aligned} \tag{2.4}$$

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

2.2. Implementaciones en *Python*

En esta sección se describen brevemente 4 implementaciones distintas para la resolución del esquema numérico 2.4 hayado en la sección anterior. Comenzamos por la más ineficiente, que correponde con el uso de la librería *NumPy*

en un esquema serial (mucho peor aún es *Python* nativo, pero vamos a saltearnos este caso), 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 de índole numérico 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 un GPU, y que pueden ser usadas por una computadora más modesta.

2.2.1. Implementación con *NumPy*

NumPy es una de las librerias 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 2.1 muestra la implementación propuesta usando *NumPy*. Nótese que hay más líneas de comentarios que de código. Esencialemente, 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 2.2 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 2.1: 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 2.2: Ejemplo de uso de la implementación con *NumPy*.

En el código 2.2 se muestra un ejemplo de uso utilizando las siguientes funciones de reacción,

$$\begin{aligned} f_u(u,v) &= -\beta uv, \\ f_v(u,v) &= \beta uv - \gamma v, \end{aligned} \tag{2.5}$$

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 2.2. 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)
```

2.2.2. Implementación serial con *Numba*

Otra de las librerías fundamentales para cálculo numérico en *Python* es *Numba*. *Numba* toma código nativo de *Python* y genera código de máquina optimizado usando *LLVM compiler infrastructure*. 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 2.3 muestra cómo sería la implementaciones 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 2.1 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 2.3: Implementación serial con *Numba*.

En cuanto al ejemplo de uso, sería idéntico al mostrado en 2.2, 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 2.1.

```
%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)
```

2.2.3. 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 2.4, pero las únicas diferencias con la versión serial 2.3 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 2.4: 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 2.3 y 10 veces más rápido que la versión de *NumPy* 2.1. Esto muestra una primera aproximación al poder de la programación en paralelo y cómo es posible implementarla sin la necesidad de una GPU que naturalemente provee una arquitectura diseñada específicamente para hacer operaciones en paralelo de manera masiva.

```
%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)
```

2.2.4. 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 2.1 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 másiva 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.

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

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 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 pudieramos 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 2.5). 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 dentendré 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 2.5: 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.

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 2.1 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 y 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 usarios sin conocimientos de *CUDA*. Sin embargo, *CuPy* ofrece también todo el acceso al detalle, tal como podría hacerse con *CUDA C* por ejemplo, y admite la escritura

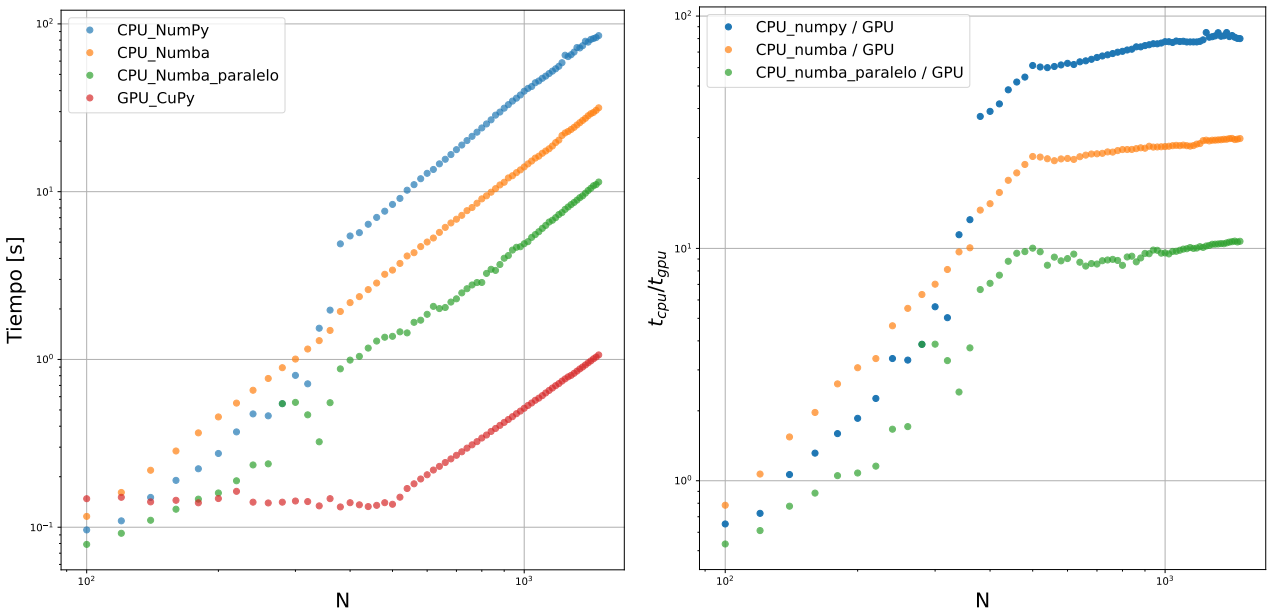

Figura 2.1: bla

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, adicionalemente, no pude obtener ninguna mejora notable respecto al *kernel* escrito con *cupy.ElementwiseKernel*. Esto no quiere decir que no se pueda, seguro que sí, estuve investigando algunas posibilidad para hacerlo, solo que no me ha dado el tiempo de explorarlas debidamente.

Capítulo 3

Simulaciones numéricas

En este capítulo presentamos los resultados obtenidos al resolver el problema propuesto en la sección ?? para los distintos casos de $\beta_{\mathbf{r}}$ utilizando métodos numéricos. Se utilizaron para ello técnicas de programación en paralelo para acelerar la velocidad de cómputo. En el apéndice A puede encontrarse la descripción de la metodología numérica utilizada, que consiste en un esquema simple de diferencias finitas sobre una grilla regular.

Todas las simulaciones realizadas en este capítulo corresponden al problema presentado en la subsección ?? pero con distintas distribuciones de $\beta_{\mathbf{r}}$. De modo que las condiciones iniciales y de contorno son las que se indican en dicha subsección, utilizando de manera general $\delta x = 1$, $L = 1024$ y $S_0 = I_0 = 1$.[1] Por otro lado, cabe mencionar que en el esquema numérico se utilizó, también de manera general, una discretización espacial del sistema de 1024×1024 sitios de igual tamaño. Los demás parámetros se especificarán según corresponda y si los mencionados aquí fueran distintos en alguna ocasión se hará mención de ello.

3.1. Medios aleatorios

En esta sección reproducimos y validamos los resultados obtenidos por A. Kolton y K. Laneri (2019) [6] en su trabajo sobre frentes de infección rugosos en medios aleatorios, donde trabajaron con la heterogeneidad dicotómica-aleatoria descrita en la sección ??. Adicionalmente, se muestran los resultados correspondientes al caso homogéneo de la sección ??. De esta manera podremos visualizar y entender la dinámica del problema heterogéneo comparando con el caso homogéneo del cual tenemos mayor entendimiento analítico.

3.1.1. Caso homogéneo

Recordemos que el problema homogéneo, donde la tasa de transmisión $\beta_{\mathbf{r}} = \beta$ es la misma en todo el espacio, es equivalente a usar la distribución ?? con $p = 0$. Se utilizó $\beta = 1$ y $\gamma = 0.2$. Se omitirán las unidades por claridad pero es importante recordar que estos valores tienen unidades de $1/t$ ya que son la tasa de infección y recuperación respectivamente[2]. Para los coeficientes de difusión se utilizó $D_S = 0$ y $D_I = 1$, los cuales deben tener dimensión de área por unidad de tiempo. Podemos ver que $\gamma/\beta = 0.2 < S_0 = 1$, de modo que según lo visto en la subsección ?? es posible la propagación de la infección a través de una solución de onda.

En la figura 3.1 se muestra la evolución del frente de infección/incendio para este caso. Puede observarse que el frente es asimétrico, es decir, se comporta de manera distinta en la parte frontal que en la parte posterior, tal como habíamos anticipado en la subsección ??. Se muestra también el campo de desplazamiento del frente $u(y, t)$ (??). Puede verse que éste es completamente plano, como es de esperar para un medio homogéneo.

[1] Se recomienda mirar nuevamente la subsección ??.

[2] De manera similar se omitirán las unidades tanto de tiempo como de longitud en la mayoría de las ocasiones. Si esto lo incomoda mucho piense que las unidades de tiempo son u_t y las de espacio u_l, poniendo β, γ, D_S y D_I en estas unidades la dinámica se describe correctamente en dichas unidades.

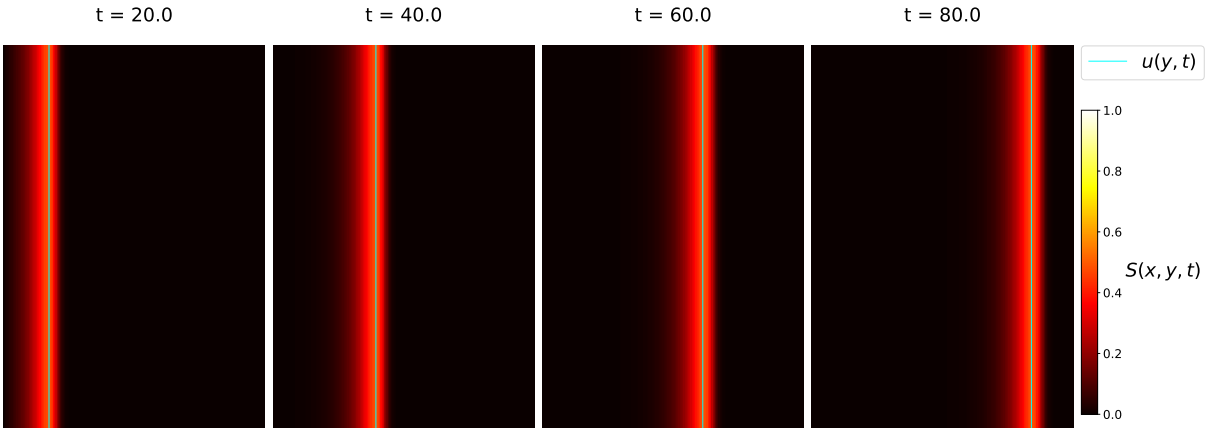

Figura 3.1: Evolución del frente de infección/incendio para el problema homogéneo.

En la figura 3.2 se muestra la posición del centro de masa del frente de propagación $u_{cm}(t)$ (??) en función del tiempo para distintos valores de γ/β. Puede observarse que para el caso $\gamma/\beta = 1$ el sistema cambia notablemente su comportamiento y no es posible identificar un frente de onda propagándose a velocidad constante, en acuerdo con la condición $\gamma/\beta < S_0 = 1$. Por otro lado, a partir de estos resultados se calculó la velocidad c (??) para cada caso a partir de un ajuste lineal descartando un periodo transitorio. De ello vemos que la velocidad del frente $c_0 = 2\sqrt{D_I\beta(S_0 - \gamma/\beta)}$ estimada en la subsección ?? concuerda con las velocidades calculadas a partir de la resolución numérica con un error relativo, en promedio, de 4.5 %. En la mayoría de los casos c_0 resulta sobreestimar levemente el valor de c, esto puede explicarse recordando que el valor de c_0 se obtuvo en una aproximación donde se supone que hay más susceptibles de los que realmente hay y por ello es de esperar que la velocidad c_0 estimada sea mayor que la real.

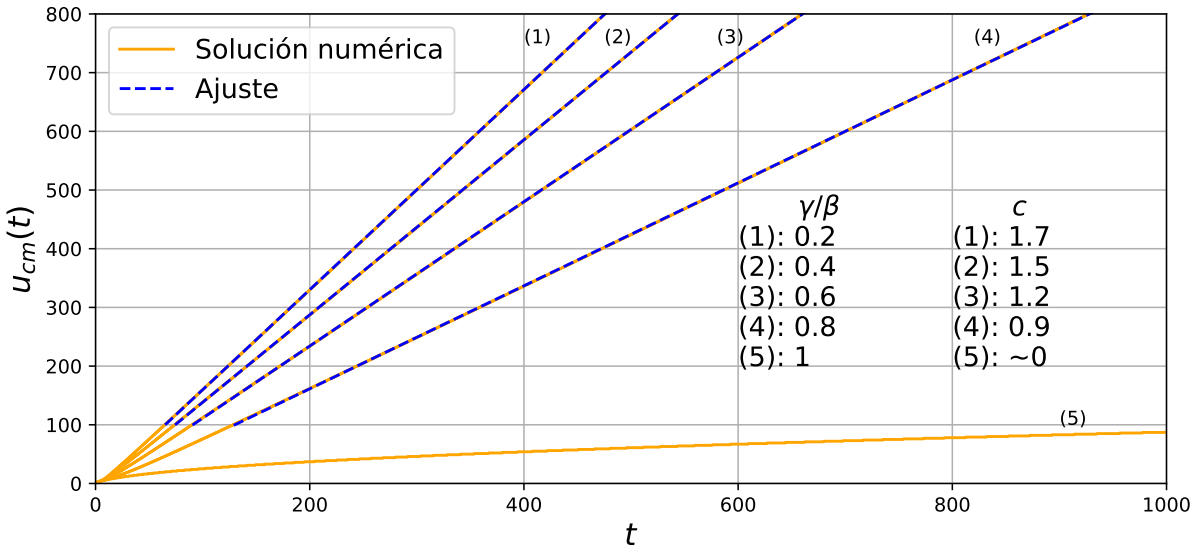

Figura 3.2: Posición del centro de masa del frente de propagación en función del tiempo para distintos valores de γ/β junto con los correspondientes ajustes lineales. Se muestra también la velocidad c del frente obtenida para cada caso.

Un observable más a destacar en esta instancia es el perfil del frente de propagación $f_I(x)$ (??), que se observa en la figura 3.3 para $\gamma/\beta = 0.2$. Se muestran también los perfiles asintóticos frontal y posterior del frente de onda calculados en la subsección ??, ecuaciones ?? y ?? respectivamente.

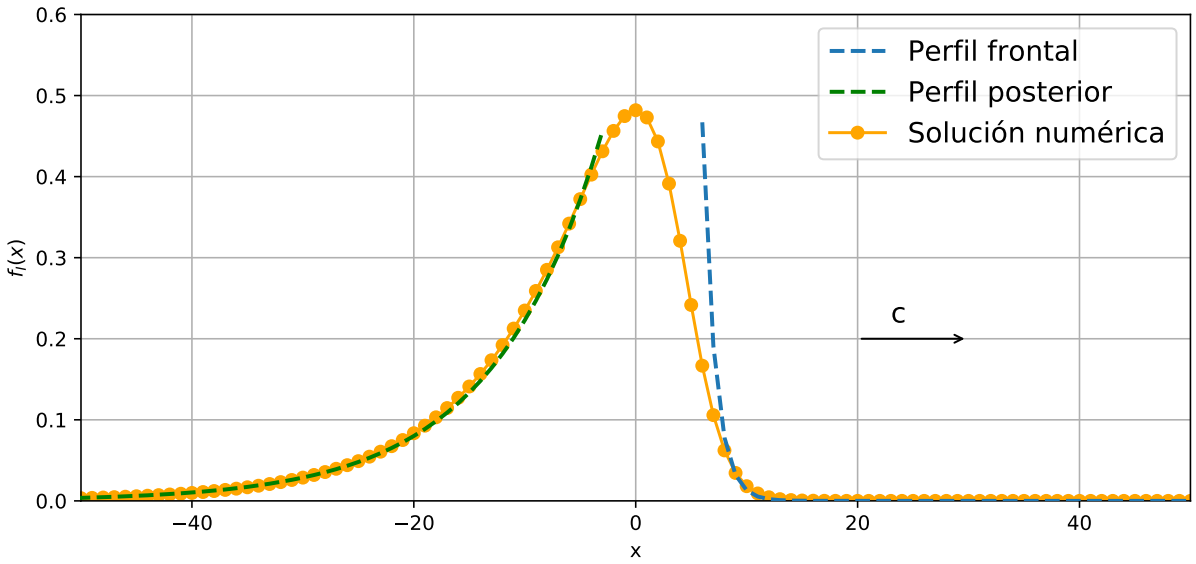

Figura 3.3: Perfil del frente de propagación $f_I(x)$ para $\gamma/\beta = 0.2$. Se muestran también los perfiles asintóticos frontal y posterior del frente de onda calculados en la sección **??**.

3.1.2. Heterogeneidad dicotómica-aleatoria

Comenzamos el estudio de la heterogeneidad dicotómica-aleatoria, con $\beta_{\mathbf{r}}$ dado por **??**, observando en la figura 3.4 la evolución temporal del frente de infección/incendio con $p = 0.3$. Puede verse el campo de desplazamiento del frente $u(y,t)$, el cual resulta rugoso en esta ocasión como consecuencia del carácter heterogéneo del medio. Vemos también que la velocidad del frente es menor respecto del caso homogéneo, como resulta claro de comparar con la figura 3.1. Este hecho no resulta sorprendente ya que el valor medio de la tasa de transmisión para este caso es $\overline{\beta_{\mathbf{r}}} = (1-p)\beta = 0.7\beta$, mientras que en el caso homogéneo es mayor (β). Lo justo sería comparar la velocidad de este caso con uno homogéneo y la misma tasa de transmisión media. Como veremos enseguida, la propagación resulta ser más rápida sobre el medio desordenado que sobre el homogéneo. Otro tema de interés es investigar cómo depende la velocidad o la amplitud media del frente con p. Recordemos que p podría representar la fracción de población vacunada, de modo que es relevante determinar cómo influye sobre la dinámica.

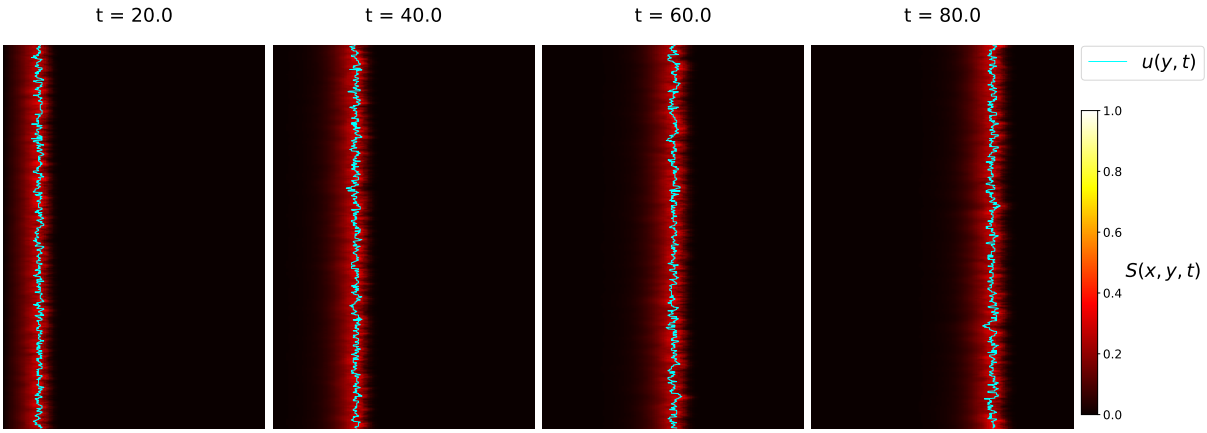

Figura 3.4: Evolución del frente de infección/incendio para el problema heterogéneo dicotómico-aleatorio con $p = 0.3$, $\gamma/\beta = 0.2$ y $D_I = 1$.

Velocidad y amplitud del frente

En la figura 3.5 se muestra la posición del centro de masa del frente de propagación $u_{cm}(t)$ en función del tiempo para realizaciones de la simulación con 70 valores distintos de $p \in (0, 0.95)$. Puede observarse que para valores de

p grandes ($p \geq 0.9$) hay un cambio de comportamiento apreciable, las trayectorias comienzan a ser inestables y la velocidad media del frente se reduce drásticamente, ya que lleva más tiempo llegar al mismo lugar. Este cambio de comportamiento como consecuencia del valor de p es de interés, ya que pareciera haber un valor crítico p_c para el que el frente de infección reduce drásticamente su velocidad a un punto tal en que es difícil incluso identificar un frente. Este hecho también puede apreciarse en la figura 3.6 donde se muestra la rugosidad del campo de desplazamiento del frente (**??**) en función de la posición del centro de masa. Para valores de $p \geq 0.9$ la rugosidad no deja de crecer mientras que para valores menores de p la rugosidad se estabiliza alrededor de un valor dado. Adicionalmente, la rugosidad es un orden de magnitud mayor en los casos en que $p \geq 0.9$ comparada con las demás, esto también habla de una disipación del frente de onda al punto en que es irreconocible.

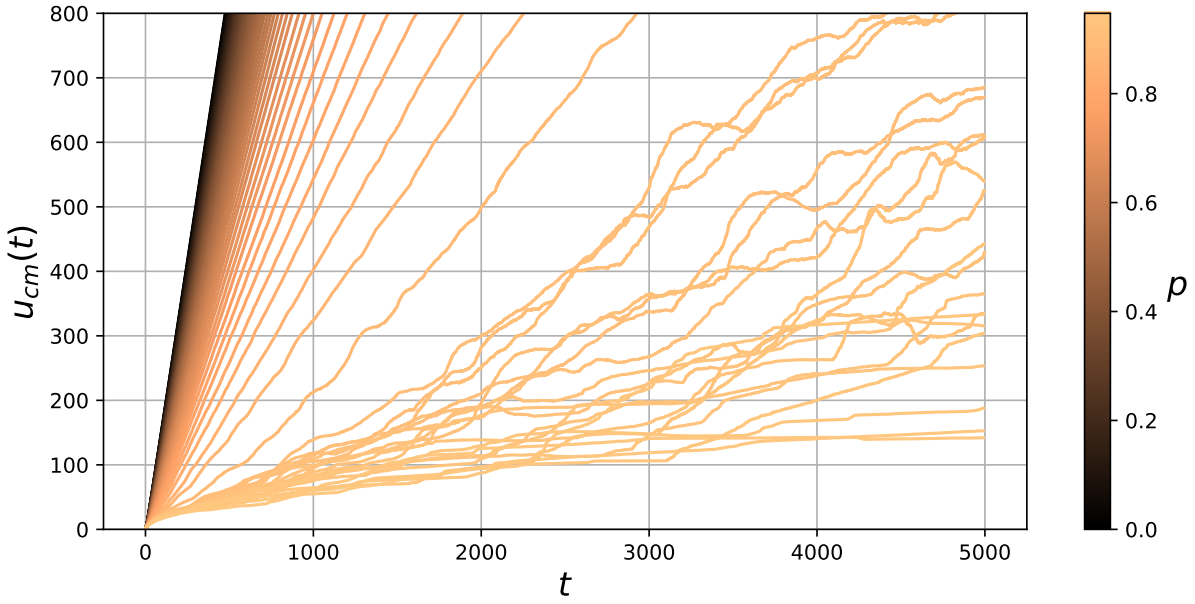

Figura 3.5: Posición del centro de masa del frente de propagación en función del tiempo para distintos valores de p,con $\gamma/\beta = 0.2$ y $D_I = 1$.

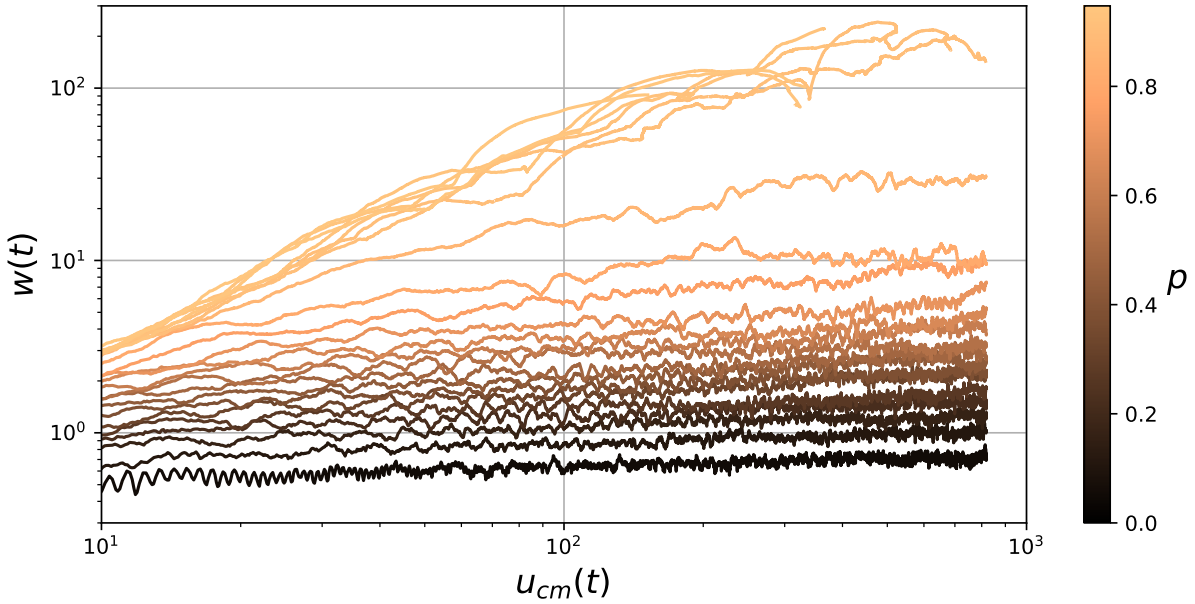

Figura 3.6: Rugosidad $w(t)$ en función de la posición del centro de masa para distintos valores de p,con $\gamma/\beta = 0.2$ y $D_I = 1$.

Para poder determinar el valor crítico p_c en el que se produce este cambio de comportamiento, calculamos a partir

Tabla 3.1: Parámetros críticos p_c y α_c de los medios H y DA con diferentes γ/β.

Medio	γ/β	p_c	α_c
DA	0.2	0.92 ± 0.04	0.61 ± 0.05
H	0.2	0.80 ± 0.01	0.47 ± 0.04
DA	0.4	0.80 ± 0.04	0.73 ± 0.05
H	0.4	0.60 ± 0.01	0.48 ± 0.04

de los resultados de la figura 3.5 la velocidad de propagación c para los distintos valores de p usando ajustes lineales. Los resultados obtenidos se muestran en la figura 3.7. Se observa este procedimiento para dos sistemas distintos con $\gamma/\beta = 0.2$ y 0.4 y $D_I = 1$. Además, se muestran estos mismos resultados para los mismos sistemas simulados sobre un medio homogéneo (H) con una tasa de transmisión igual a la media de sus pares heterogéneos dicotómicos-aleatorios (DA). Vemos que en ambos casos la velocidad del frente es menor sobre el medio homogéneo que sobre el heterogéneo y la diferencia entre ambas velocidades incrementa con p. Este tipo de resultado no es trivial y como es propio de los sistemas complejos no siempre es posible desarrollar una intuición apropiada para describir fenómenos emergentes de este tipo. Es decir, con la misma tasa de transmisión media, una distribución homogénea sobre el espacio da lugar a un frente de propagación de menor velocidad que una distribución heterogénea dicotómica y aleatoria.

Se realizó un ajuste con la regla de potencia $c \propto (1 - p/p_c)^{\alpha_c}$ para las realizaciones sobre el medio homogéneo en todo el rango de p. De lo cual se obtuvo $p_c \approx 0.8$ y 0.6 para $\gamma/\beta = 0.2$ y 0.4 respectivamente con exponente crítico $\alpha_c \approx 0.5$ para ambos. Esto concuerda con lo que resulta de reemplazar β por $(1-p)\beta$ en la ecuación para la velocidad sobre medios homogéneos determinada en **??**, es decir,

$$c(p) = 2\sqrt{D_I((1-p)\beta S_0 - \gamma)} = c_0(1 - p/p_c)^{\alpha_c},$$

con $p_c = 1 - \gamma/(\beta S_0)$ y $\alpha_c = 0.5$.

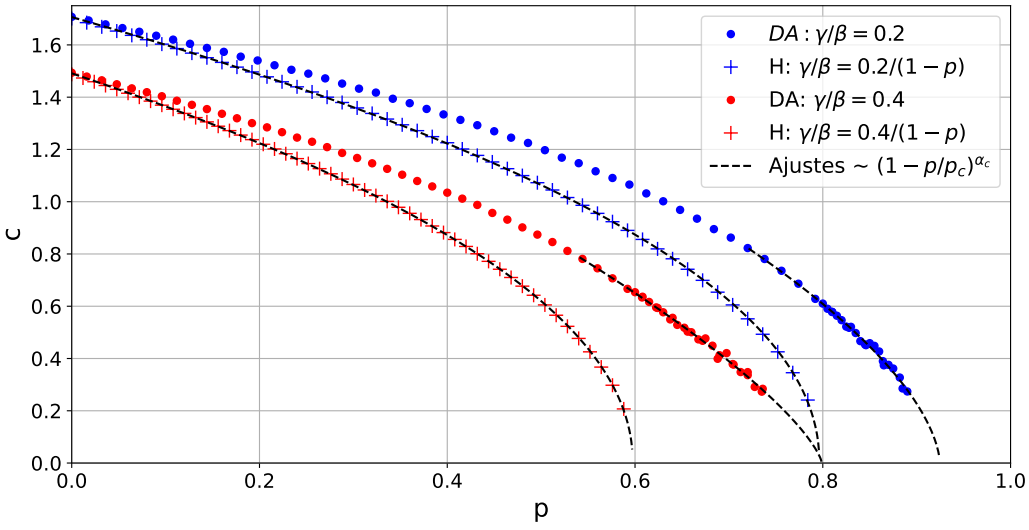

Figura 3.7: Velocidad c del frente de propagación en función de p con $\gamma/\beta = 0.2$ y 0.4 y $D_I = 1$. Se muestran los ajustes con la regla de potencia $c \propto (1 - p/p_c)^{\alpha_c}$ sobre la región crítica.

Por otro lado, para los medios DA se realizó un ajuste similar sobre las regiones críticas de p. Todos los resultados asociados a los parámetros críticos de la velocidad determinados para cada caso se muestran en la tabla 3.1.

De estos resultados queda claro otra diferencia importante entre los casos H y DA, y es que el valor crítico de p para el cual se extingue el frente de propagación es apreciablemente mayor en los casos heterogéneos, hasta un 30 % mayor en el caso de $\gamma/\beta = 0.4$.

Es posible realizar un procedimiento similar determinando la amplitud media del frente de infección I_{max} (**??**)

para distintas realizaciones variando el valor de p. En la figura 3.8 se muestran los resultados obtenidos. En este caso vemos que sale desfavorecido el medio homogéneo ya que la amplitud media I_{max} del frente de infección resulta ser mayor que en el medio aleatorio. Se realizó asimismo un ajuste con la regla $(1-p/p_c)^{\alpha_I}$ para los casos heterogéneos sobre las regiones críticas, en la tabla 3.2 se muestran los resultados obtenidos.

Tabla 3.2: Parámetros críticos p_c y α_I del medio DA con diferentes γ/β.

Medio	γ/β	p_c	α_I
DA	0.2	0.90 ± 0.04	1.93 ± 0.05
DA	0.4	0.75 ± 0.04	2.12 ± 0.05

Puede verse que dentro del error, los resultados obtenidos para p_c concuerdan con los obtenidos del ajuste para la velocidad. Por otro lado, es posible observar que tanto para el caso de la amplitud media como de la velocidad los exponentes críticos difieren del dado en los casos homogéneos y también difieren entre sí.

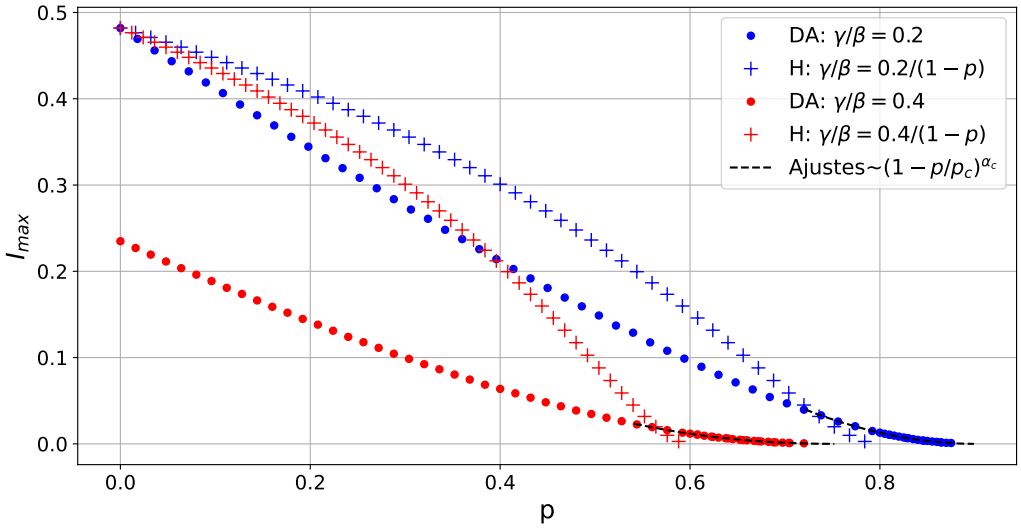

Figura 3.8: Amplitud media I_{max} del frente de propagación en función de p con $\gamma/\beta = 0.2$ y 0.4 y $D_I = 1$. Se muestran los ajustes con la regla de potencia $I_{max} \propto (1-p/p_c)^{\alpha_I}$ sobre la región crítica.

Para ver más resultados asociados a esta heterogeneidad se puede consultar el artículo [6]. Allí se presentan resultados adicionales acerca del factor de estructura del frente $S(q)$ (**??**) y del perfil del frente $f_I(x)$. Se encuentra una estructura auto-afín del frente sobre la región crítica, con una ley del tipo $S(q) \propto 1/q^{1+2\zeta}$.

En resumen, los resultados obtenidos aquí pueden describirse cualitativamente de la siguiente manera:

- Si el valor medio de la tasa de transmisión $\overline{\beta_{\mathbf{r}}}$ es el mismo, la velocidad del frente de infección/incendio es *mayor* sobre el medio dicotómico-aleatorio que sobre el homogéneo.
- Si el valor medio de la tasa de transmisión $\overline{\beta_{\mathbf{r}}}$ es el mismo, la amplitud media del frente de infección/incendio I_{max} es *mayor* sobre el medio homogéneo que sobre el dicotómico-aleatorio en tanto $p < p_c$, con p_c del medio homogéneo.
- El valor crítico p_c para el cual se extingue el frente de infección/incendio es *mayor* sobre el medio dicotómico-aleatorio que sobre el homogéneo.
- Los exponentes críticos de la velocidad α_c para el medio homogéneo *coinciden* con el estimado por c_0 en la aproximación analítica de la solución de onda.
- Los exponentes críticos de la velocidad α_c para el medio dicotómico-aleatorio son *mayores* que para el medio homogéneo y parecen *variar* con γ/β.

3.2. Medios correlacionados

En esta sección estudiamos las heterogeneidades con correlación introducidas en la sección **??**, estas son las denominadas «suavizada» (S) y «dicotómica-correlacionada» (DC).

En las figuras 3.9 y 3.10 se muestra la evolución temporal del sistema para $p = 0.3$ y con un paso de suavizado para cada caso, es decir, con $\beta_{\mathbf{r}}^{(1)}$ y $\tilde{\beta}_{\mathbf{r}}^{(1)}$ respectivamente. Puede verse que DC presenta un campo de desplazamiento más rugoso que S y que la velocidad del frente en S es mayor que en DC. Sin embargo, aquí nuevamente resulta que la comparación es injusta ya que, aunque ambas simulaciones están hechas con el mismo valor de p, tienen distinta tasa de transmisión media. Como se vió en la sección **??**, el valor medio sobre S es $\overline{\beta_{\mathbf{r}}^{(n)}} = (1-p)\beta$ mientras que sobre DC el valor medio de la tasa de transmisión cambia. Por ello utilizaremos de ahora en más como parámetro de referencia el valor medio de la tasa de transmisión y no p.

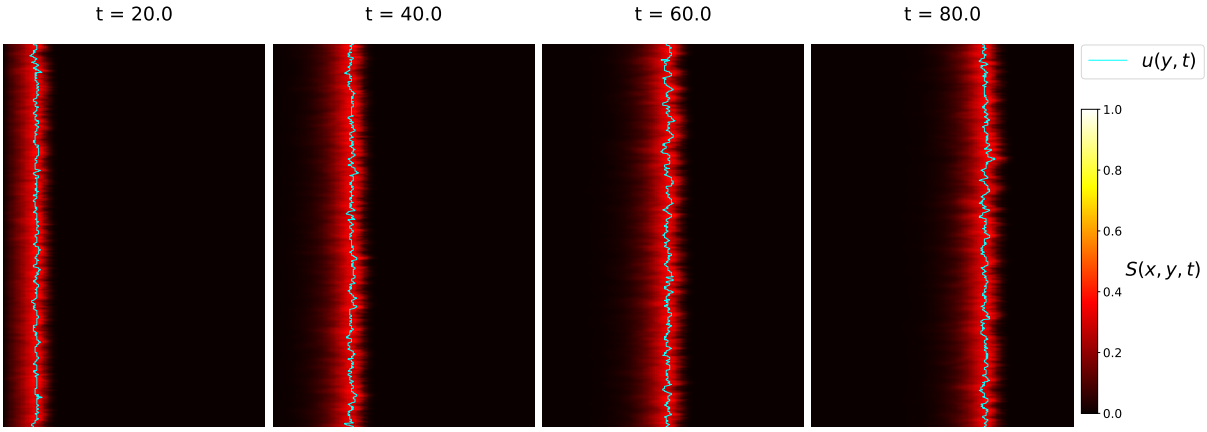

Figura 3.9: Evolución temporal del sistema para $p = 0.3$ con un paso de S.

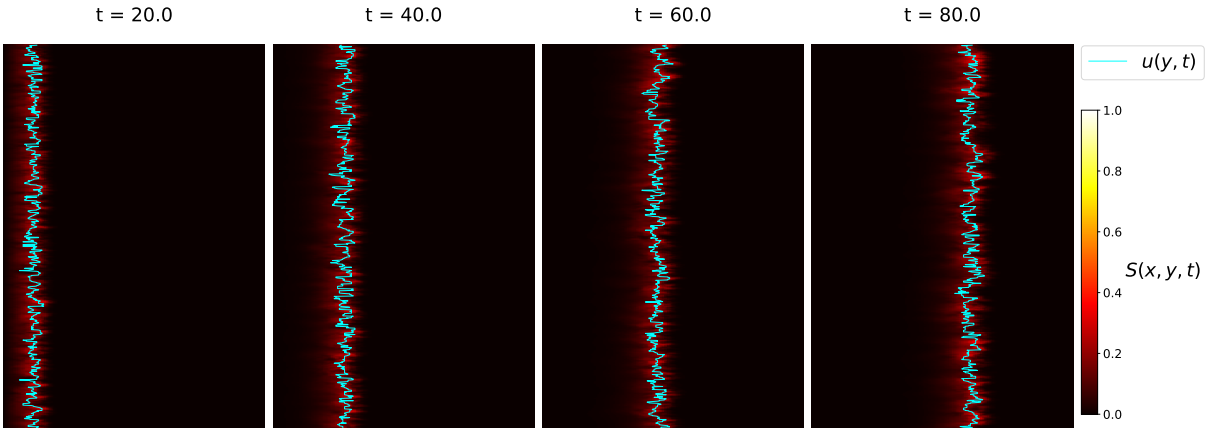

Figura 3.10: Evolución temporal del sistema para $p = 0.3$ con un paso de DC.

3.2.1. Velocidad y amplitud del frente

En las figuras 3.11 y 3.12 se muestra la posición del centro de masa del frente de realizaciones con distinta tasa de transmisión media, para S y DC respectivamente. Se utilizó $\gamma = 0.2$, $\beta = 1$ y $D_I = 1$.

Se puede apreciar, en ambos casos, que mientras mayor es la tasa de transmisión media, mayor es la velocidad del frente de infección/incendio, esto es algo de esperar. Sin embargo, algo que llama la atención es que el frente de infección parece sostener una velocidad mayor sobre DC que sobre S a medida que disminuye $\overline{\beta_{\mathbf{r}}}$.

Lo mencionado anteriormente queda claro al observar la figura 3.13, donde se muestra la velocidad en función de $\overline{\beta_{\mathbf{r}}}$ sobre los medios S y DC con un paso de suavizado. Adicionalmente, se muestran las curvas equivalentes para los medios homogéneo (H) y dicotómico-aleatorio (DA), vistos en 3.1. Vemos que la curva asociada a S queda entre las

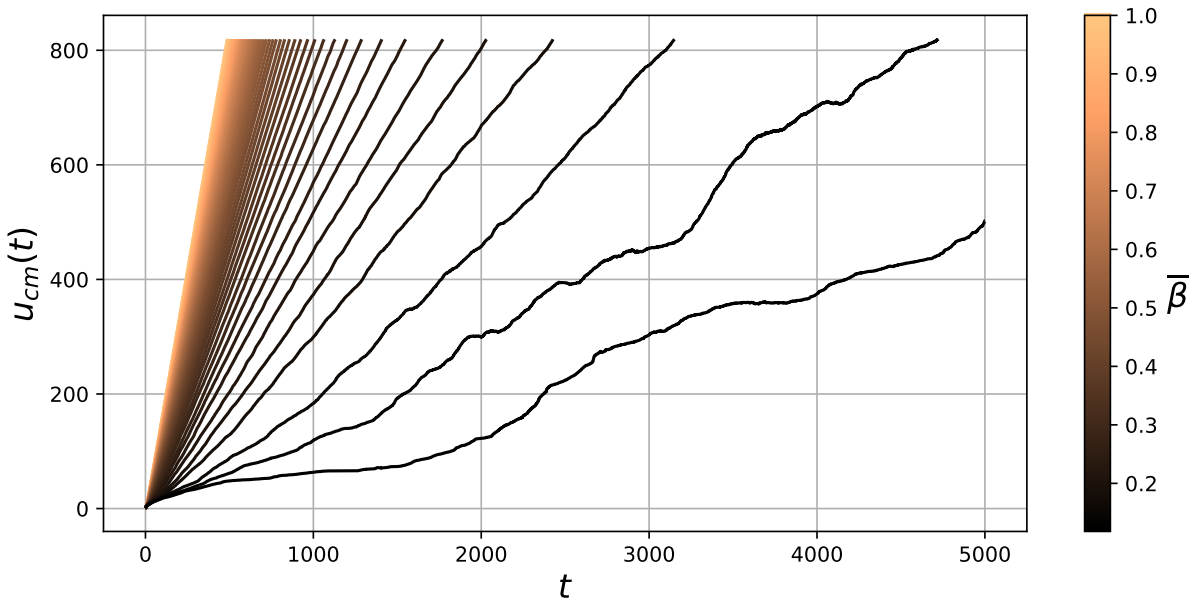

Figura 3.11: Posición del centro de masa del frente en función del tiempo sobre el medio S.

curvas de los medios H y DA. Esto es razonable ya que S se genera a partir de DA y como vimos en la sección **??**, el medio S tiende a uno homogéneo con $\overline{\beta_{\mathbf{r}}} = (1-p)\beta$ cuando incrementamos los pasos de suavizado. Es de esperar entonces que curvas similares sobre S pero con más pasos de suavizado vayan asemejándose cada vez más a la versión homogénea.

Por otro lado, la curva asociada a DC queda alejada de las demás, se observa que la velocidad del frente es apreciablemente más grande que en los demás casos cuando la tasa de transmisión media es chica, $\overline{\beta_{\mathbf{r}}} < 0.2$. Ahora bien, este resultado por sí mismo es por lo menos curioso, es decir, **un ligero cambio en la estructura del medio sin afectar la tasa de transmisión media produce un cambio notable en la velocidad del frente**. Fundamentalmente, comparando el resultado de DC con el de DA, a pesar de que ambos medios tengan un carácter dicotómico, la correlación espacial introducida en DC genera un cambio notable sobre la velocidad.

Para entender mejor qué es lo que produce este cambio de velocidad entre DA y DC es interesante observar con mayor detenimiento la estructura de estos medios. En las figuras 3.14 y 3.15 se muestra la distribución de la tasa de transmisión $\beta_{\mathbf{r}}$ sobre el espacio de los medios DA y DC respectivamente. Ambos casos tienen la misma tasa de transmisión media. Observando la distribución espacial de cada uno de ellos a gran escala, sobre el espacio de 1024×1024, no es sencillo señalar ninguna diferencia. Sin embargo, al observar la distribución sobre una región acotada de 100×100 se puede ver que son estructuralmente distintos en esta escala. La distribución de CD con $n = 1$ alcanza a formar un patrón distinguible, se forman regiones agrupadas con tasa de transmisión no nula β, mientras que en DA, por supuesto, se observa el carácter aleatorio. Este es un fenómeno notable, la velocidad de propagación es mayor si la distribución de la tasa de transmisión forma estructuras como las que se ven en la figura 3.15 en lugar de estar distribuída aleatoriamente, por más que la tasa de transmisión media sea la misma.

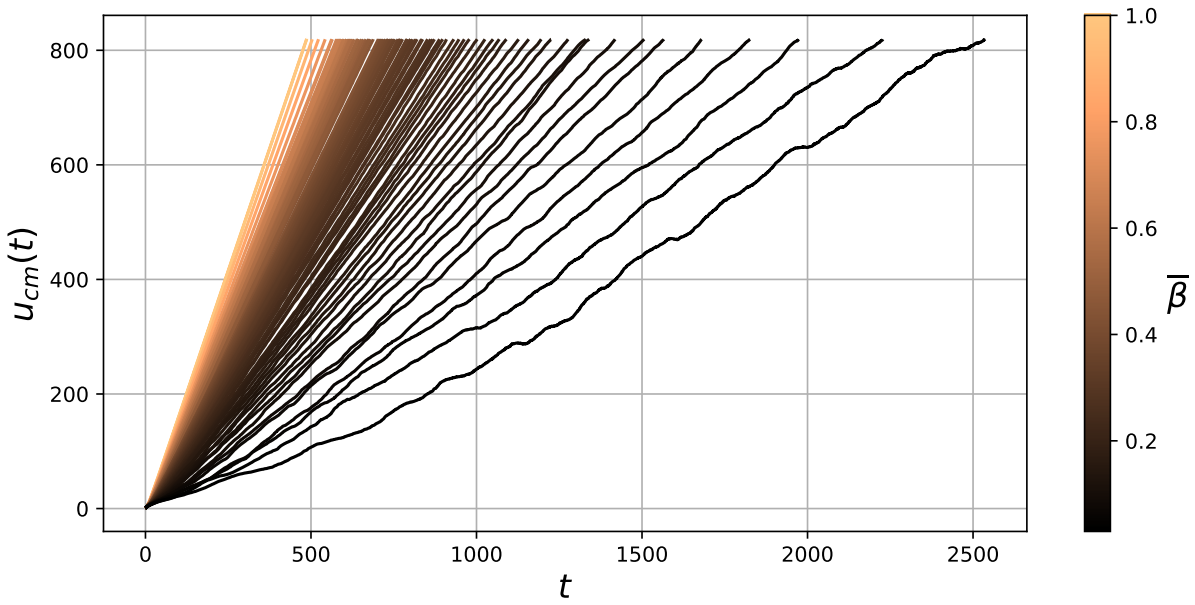

Figura 3.12: Posición del centro de masa del frente en función del tiempo sobre el medio DC.

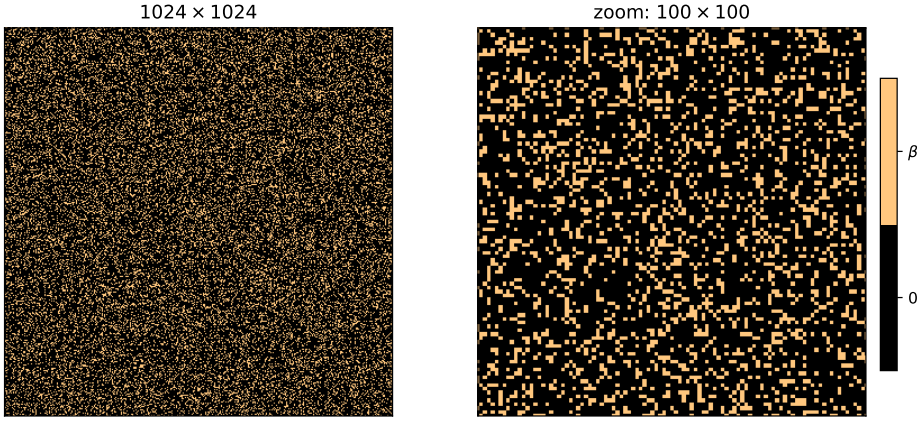

Figura 3.14: Distribución de la tasa de transmisión $\beta_{\mathbf{r}}$ del medio DA. A izquierda se muestra la distribución sobre todo el espacio de 1024×1024 y a la derecha un acercamiento a una región de 100×100.

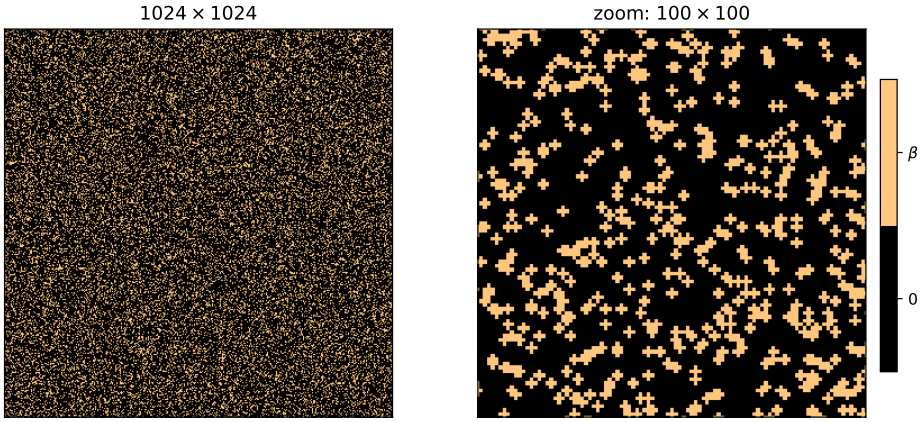

Figura 3.15: Distribución de la tasa de transmisión $\beta_{\mathbf{r}}$ del medio DC. A izquierda se muestra la distribución sobre todo el espacio de 1024×1024 y a la derecha un acercamiento a una región de 100×100.

Se realizaron ajustes con la regla de potencia $c \propto (\beta - \beta_c)^{\alpha_c}$ sobre la región crítica para los medios DC, DA, S y

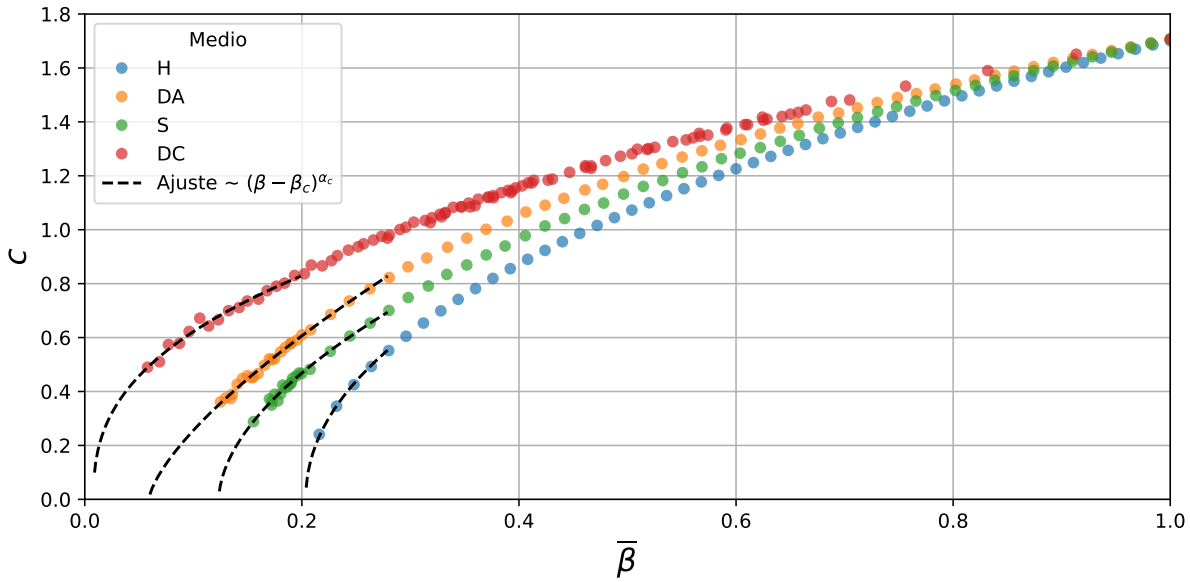

Figura 3.13: Velocidad del frente de infección en función del valor medio de la tasa de transmisión $\overline{\beta_{\mathbf{r}}}$ sobre los medios S, DC, DA y H. Se muestran los ajustes con la regla de potencia $c \propto (\beta - \beta_c)^{\alpha_c}$ sobre la región crítica.

H. De ello se determinó la tasa de transmisión crítica y el exponente crítico para cada uno de ellos. En la tabla 3.3 se muestran los resultados obtenidos.

Tabla 3.3: Parámetros críticos β_c y α_c de los medios DC, DA ,S y H.

Medio	β_c	α_c
H	0.20 ± 0.01	0.48 ± 0.03
DA	0.06 ± 0.02	0.70 ± 0.05
DC	0.01 ± 0.01	0.39 ± 0.05
S	0.12 ± 0.02	0.55 ± 0.05

Puede verse que los resultados sobre los medios H y DA coinciden con los valores críticos p_c obtenidos en la sección anterior (Tabla 3.1). Esto era de esperar dado que sobre estos medios, la tasa de transmisión media y el valor de p están relacionados por $\overline{\beta_{\mathbf{r}}} = (1-p)\beta$, de modo que $\beta_c = (1-p_c)\beta$. De estos resultados también queda claro otro fenómeno notable al que da lugar la estructura del medio DC, y es que sobre este la tasa de transmisión crítica es *menor* comparada con las demás. Es decir, basta con una tasa de transmisión extremadamente chica, comparada con los demás medios, para que tenga lugar un frente de propagación.

Otro aspecto de inteŕes es observar el efecto que tienen los distintos medios sobre la amplitud media del frente de infección/incendio . En la figura 3.16 se observa la amplitud media del frente como función de la tasa de transmisión media $\overline{\beta_{\mathbf{r}}}$ para los medios S, DC, DA y H. En esta ocasión vemos que el medio homogéneo es el que sostiene la mayor amplitud del frente cuando $\overline{\beta_{\mathbf{r}}} > 0.35$. Vemos aquí nuevamente que la curva de S queda entre las dadas por H y DA. Por otro lado, el medio DC, en consistencia con lo observado para la velocidad, otorga la mayor amplitud del frente para valores bajos de la tasa de transmisión media $\overline{\beta_{\mathbf{r}}} < 0.35$. Es decir, el medio DC favorece nuevamente, en este aspecto y sobre este régimen, a la propagación del frente.

3.2.2. Rugosidad y factor de estructura

Ahora se propone investigar las propiedades geométricas del frente en función de $\overline{\beta_{\mathbf{r}}}$ cerca de los valores críticos β_c obtenidos previamente para cada medio. En la figura 3.17 se muestra la rugosidad $\langle w(t) \rangle_t$ del frente de infección/incendio en función de la tasa de transmisión media para realizaciones sobre los medios S, DA y DC.

Para cada una de estas curvas se realizó un ajuste de tipo $\langle w(t) \rangle_t \propto (\beta - \beta_c)^{\alpha_c}$ sobre la región crítica. A partir de lo cual se obtuvieron los resultados que se muestran en la tabla 3.4 para cada uno de los medios.

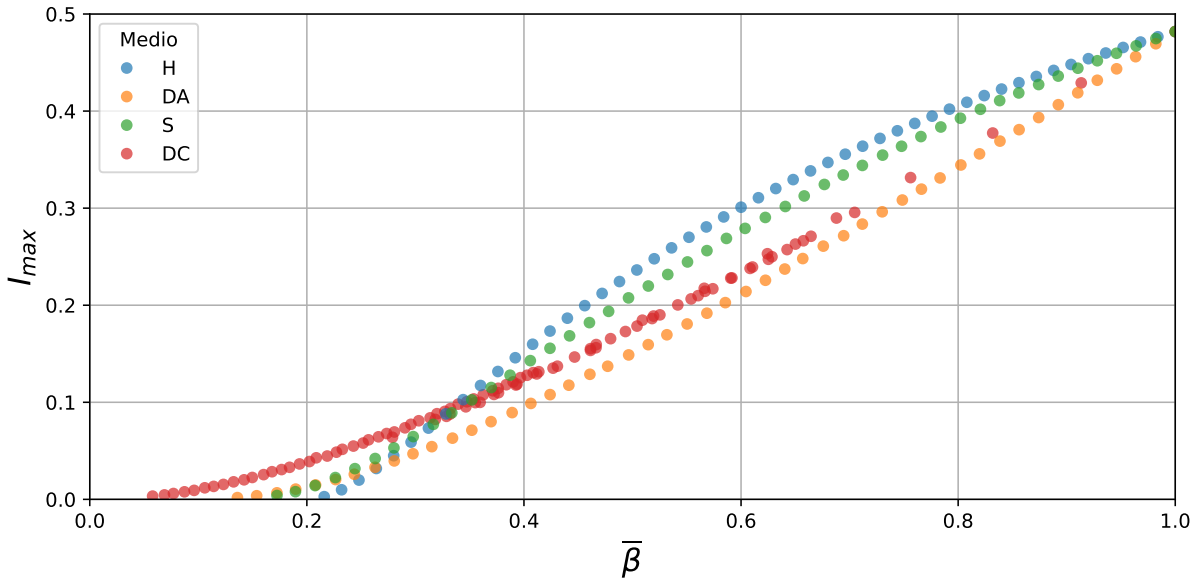

Figura 3.16: Amplitud media del frente de infección/incendio I_{max} en función de la tasa de transmisión media $\overline{\beta_{\mathbf{r}}}$ para los medios S, DA, DC y H.

Tabla 3.4: Parámetros críticos β_c y α_w de los medios DC, DA y S.

Medio	β_c	α_w
DA	0.07 ± 0.02	1.61 ± 0.05
DC	0.01 ± 0.01	0.96 ± 0.05
S	0.11 ± 0.02	1.48 ± 0.05

Los valores críticos de β_c concuerdan con los obtenidos previamente del estudio de la velocidad del frente, tal como corresponde (Tabla 3.3).

Por otro lado, si observamos el factor de estructura $S(q)$ (**??**) de los frentes cerca del valor crítico de la tasa de transmisión media, se observa que sobre el medio DA el frente presenta una estructura fractal auto-afín, $S(q) \sim 1/q^{1+2\zeta}$, con exponente de rugosidad $\zeta = 0.3$ tal como puede verse de la figura 3.18. Sin embargo, para los medios S y DC no se observa nada similar, los valores de ζ son indistinguibles de cero.

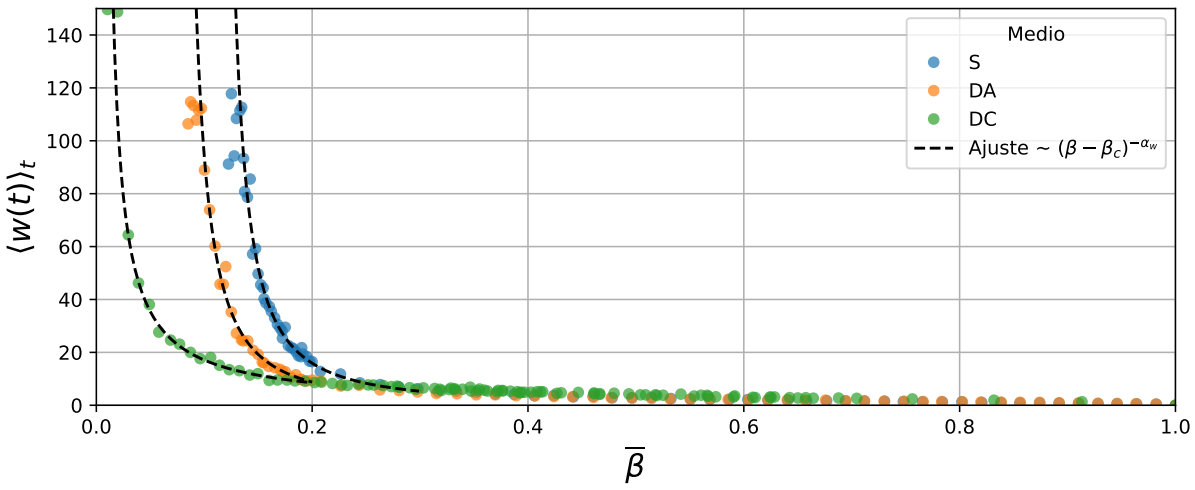

Figura 3.17: Rugosidad del frente de infección/incendio en función de la tasa de transmisión media $\overline{\beta_{\mathbf{r}}}$ para los medios S, DA y DC.

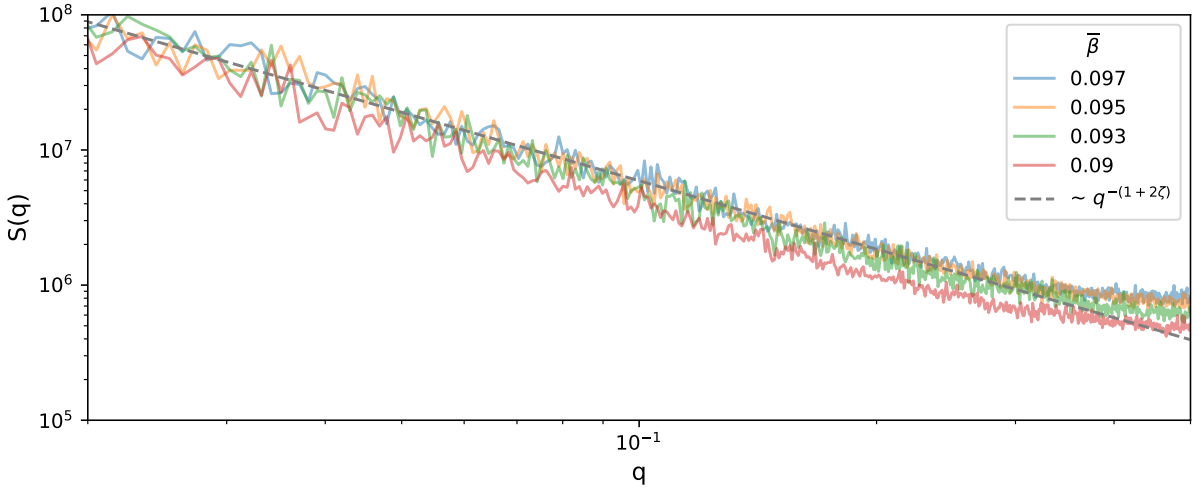

Figura 3.18: Factor de estructura $S(q)$ sobre el medio DA cerca de la tasa de transmisión crítica. Se muestra el ajuste $S(q) \sim 1/q^{1+2\zeta}$ con $\zeta \approx 0.3$.

Resumiendo esta sección, se tienen los siguientes resultados:

- Si la tasa de transmisión media es la misma, la velocidad del frente de infección/incendio es *mayor* sobre el medio DC.
- Si la tasa de transmisión media es la misma y $\beta_{\mathbf{r}} > 0.35$, la amplitud media del frente de infección/incendio es *mayor* sobre el medio H.
- Si la tasa de transmisión media es la misma y $\beta_{\mathbf{r}} < 0.35$, la amplitud media del frente de infección/incendio es *mayor* sobre el medio DC.
- La tasa de transmisión crítica es la *menor* sobre el medio DC y la *mayor* sobre el medio H.
- Los exponentes críticos de la velocidad sobre los medios H, S, DA y DC parecen ser *diferentes* entre sí, siendo el *mayor* en el medio DA.
- El campo de desplazamiento del frente presenta una estructura auto-afín en el medio DA sobre el régimen crítico, $S(q) \propto q^{-(1+2\zeta)}$ con $\zeta = 0.3$.

3.3. Nocividad

Un aspecto de gran importancia en lo que respecta a problemáticas tanto epidemiológicas como de incendios es la posibilidad de cuantificar los daños que deja un brote de enfermedad infecciosa o un incendio. En particular, estaremos interesados en estimar el daño que queda tras el paso del frente de infección/incendio. Para ello utilizaremos como cuantificador de daños la fracción de susceptibles S_1 que queda tras el paso del frente de onda. Entendiendo que mientras menor sea esta fracción, mayor será el daño ocasionado por el frente.

Para ser más precisos al respecto, tomaremos

$$S_1 = \langle S(x, y, T) \rangle_{x,y}$$

con $0.2L < x < u_{cm}(T) - 0.2L$, siendo T el tiempo que dure la simulación correspondiente. Utilizando este criterio para todas las simulaciones, podremos comparar la nocividad de cada frente al modificar los parámetros que interesen. En particular, veremos cómo cambia esta magnitud S_1 con la tasa de transmisión media $\overline{\beta_{\mathbf{r}}}$.

En la figura 3.19 se muestran los resultados obtenidos de S_1 en función de $\overline{\beta_{\mathbf{r}}}$ sobre los medios H, S, DA y DC. Se puede observar que el frente de propagación más nocivo se da sobre el medio homogéneo, ya que da lugar a la menor fracción de susceptibles S_1 tras el paso del frente. Sin embargo, tal como ya hemos apreciado antes (Tabla 3.3), la tasa de transmisión crítica es menor que en los demás casos $\beta_c \approx 0.2$.

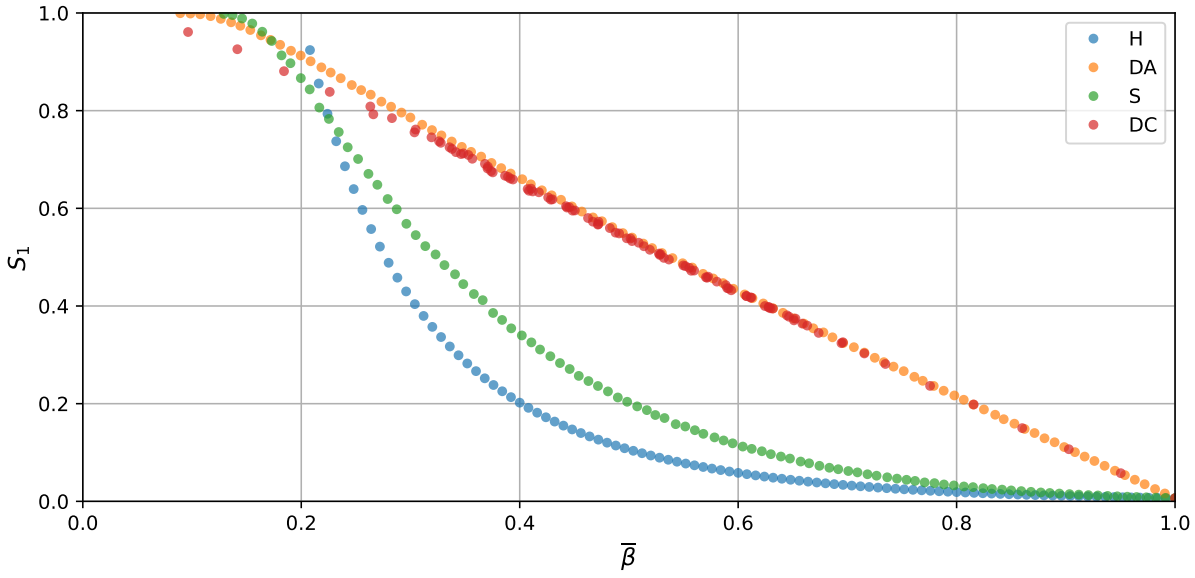

Figura 3.19: Fracción de susceptibles S_1 en función de la tasa de transmisión media $\overline{\beta_{\mathbf{r}}}$ para los medios H, S, DA y DC.

El siguiente medio de nocividad elevada es S, mientras que los medios DC y DA presentan curvas similares de carácter lineal. Estos nuevos resultados dan una perspectiva nueva acerca de los medios. Por ejemplo, habíamos visto que la velocidad de propagación del frente es mayor sobre el medio DC que sobre los demás, lo cual es una característica indeseable en problemáticas tanto epidemiológicas como de incendios. Sin embargo, vemos ahora que a pesar de ello la nocividad del frente, tal como la hemos caracterizado aquí, es menor sobre DC que sobre los demás medios, lo cual es deseable. A modo cualitativo es entendible que si el frente se desplaza a mayor velocidad tenga menos tiempo de ocasionar grandes daños.

En resumen:

- Si $\overline{\beta_{\mathbf{r}}} > 0.2$, el frente de infección/incendio *más nocivo* se da sobre el medio H.
- Si $\overline{\beta_{\mathbf{r}}} < 0.2$, el frente de infección/incendio *más nocivo* se da sobre el medio DC.
- Sobre los medios dicotómicos DC y DA, la *nocividad* es directamente proporcional a la tasa de transmisión media, $S_1 \propto \overline{\beta_{\mathbf{r}}}$.

Capítulo 4

Conclusiones

Se utilizaron herramientas estadísticas y computacionales para caracterizar frentes de onda gobernados por ecuaciones de reacción-difusión asociadas al modelo SIR sobre una variedad de medios estadísticamente isotrópicos. Estos frentes de onda son característicos de diversas fenomenologías asociadas a ecuaciones de reacción-difusión.

En particular, en el ámbito epidemiológico, modelar el efecto que tiene el carácter heterogéneo de la distribución espacial de las poblaciones sobre la dinámica infecciosa constituye un desafío complejo y sin lugar a dudas moderno.[2]

El trabajo desarrollado aquí constituye un paso en esa dirección. Fue posible obtener resultados no triviales respecto de la influencia del medio sobre los frentes de propagación. Características de interés como la velocidad de propagación, la amplitud media, la rugosidad del campo de desplazamiento e incluso la nocividad de los frentes fueron desarrolladas a nivel cuantitativo. Exponentes críticos asociados a un cambio radical de la dinámica fueron determinados sobre los distintos medios.

Por otro lado, en el modelado de incendios forestales es claro que la topografía, el carácter heterogéneo del medio y las condiciones climáticas son factores determinantes. Estos podrían introducirse en el modelo a partir de una interpretación precisa de los efectos que las variaciones en el medio tienen sobre la dinámica y ser estudiados utilizando las mismas herramientas desarrolladas aquí.

Es importante notar que exponentes críticos tal como los determinados en este trabajo, suelen estar asociados a una dada clase de universalidad, que resulta independiente de los detalles propios del problema estudiado aquí. [9] Esto daría la posibilidad de estudiar sistemas más complejos dentro de la misma clase de universalidad a partir del modelo más simple que se ha explorado en este trabajo.

Para futuros trabajos en esta línea sería interesante investigar los efectos de la dinámica sobre medios hiperuniformes, tanto ordenados como desordenados. Así como medios con configuraciones periódicas o bien tan arbitrarias como se quiera. También se propone trabajar con medios dinámicos y no solo estacionarios como los estudiados aquí.

Apéndice A

Metodología numérica

El sistema de ecuaciones dado por

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

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

se resolvió numéricamente utilizando el siguiente esquema explícito de Euler:

$$S_{ij}^{n+1} = S_{ij}^n + \Delta t \left(-\beta_{ij} S_{ij}^n I_{ij}^n + \frac{D_S}{d^2} (S_{i+1j}^n + S_{i-1j}^n + S_{ij+1}^n + S_{ij-1}^n - 4S_{ij}^n) \right)$$

$$I_{ij}^{n+1} = I_{ij}^n + \Delta t \left(\beta_{ij} S_{ij}^n I_{ij}^n - \gamma I_{ij}^n + \frac{D_I}{d^2} (I_{i+1j}^n + I_{i-1j}^n + I_{ij+1}^n + I_{ij-1}^n - 4I_{ij}^n) \right)$$

donde los índices i y j corresponden a la discretización de las coordenadas espaciales y el índice n corresponde a la discretización temporal. Se utilizaron de manera general los siguientes parámetros $\Delta t = 0.1$, $d = 1$, $D_I = 1$ y $D_S = 0$. Los demás parámetros involucrados se indican según corresponde en el texto.

Este esquema se implementó por computación en paralelo utilizando procesadores gráficos. De esta manera fue posible resolver de manera eficiente cientos de sistemas sobre grillas de 1024×1024. En particular se utilizó la librería de `Python` para computación en paralelo llamada `CuPy`, que es un equivalente de la librería `NumPy` pero integrada con `CUDA`.

A continuación se muestra la función central del esquema numérico que permite recorrer la grilla del sistema en paralelo y calcular las derivadas del sistema en cada nodo utilizando esta librería, se representan a S e I como X e Y respectivamente:

```
forces = cp.ElementwiseKernel(
'raw float64 X, raw float64 Y, raw float64 params,raw float64 beta,raw float64 gamma, int16 L' ,
'float64 fX,float64 fY',
'''
double N = params[0]; double nu = params[1]; double mu = params[2]; double Dx = params[3];
double Dy = params[4];

int x = i % L;
int y = (int) i/L;

fX =  nu - beta[i]*X[i]*Y[i]/N - mu*X[i] +
    Dx*(X[(x+1)%L + L*y] + X[(x-1+L)%L+L*y] + X[x + L*((y+1)%L)] + X[x + L*((y-1+L)%L)] - 4*X[i]);

fY = beta[i]*X[i]*Y[i]/N - (gamma[i]+mu)*Y[i] +
    Dy*(Y[(x+1)%L + L*y] + Y[(x-1+L)%L+L*y] + Y[x + L*((y+1)%L)] + Y[x + L*((y-1+L)%L)] - 4*Y[i]);
''',
'forces')
```

y el paso de Euler se lee simplemente como:

```
...
forces(X,Y,params,beta,gamma,Lx,fX,fY)
X = X + tstep*fX
Y = Y + tstep*fY
...
```

Para dar una idea de la aceleración dada por la programación en paralelo, en la figura A.1 se muestra el tiempo necesario para resolver un sistema de $N \times N$ sitios y 1000 pasos de Euler con procesadores gráficos (GPU) y con procesadores convencionales (CPU). La diferencia es notable, por ejemplo, para un sistema de 1024×1024, el tiempo necesario para resolverlo con CPU es aproximadamente de 2.7 horas, mientras que con GPU se resuelve en 1 segundo. Se muestra también un ajuste para ambas curvas del tipo $t \propto N^a$ con $a \approx 2$ para ambos. De todo esto resulta clara la necesidad de trabajar con computación en paralelo para obtener los resultados desarrollados en este trabajo en un tiempo razonable. Para ver más detalles acerca del código y las herramientas computacionales implementadas puede acceder al siguiente *Notebook* de *Google Colab*.

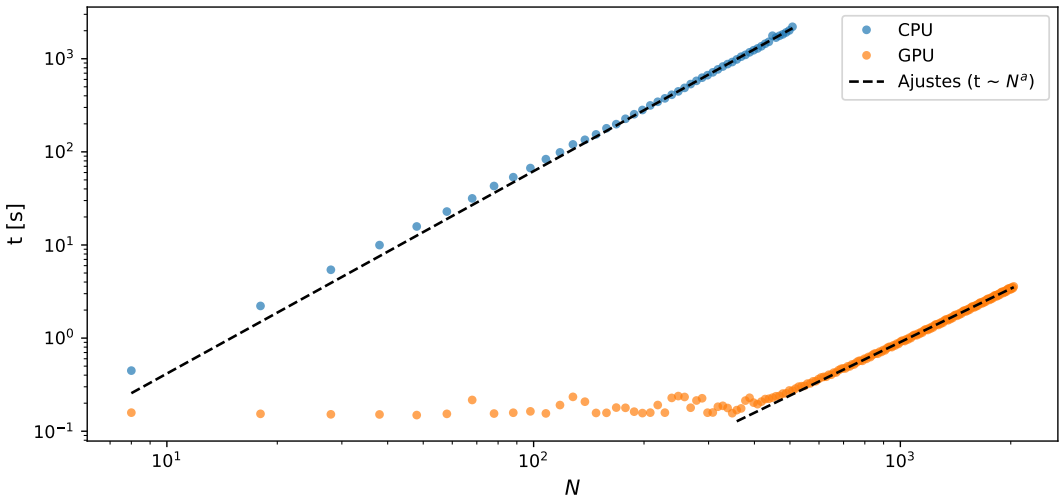

Figura A.1: Tiempo de resolución de un sistema de $N \times N$ sitios y 1000 pasos de Euler con procesadores gráficos (GPU) y con procesadores convencionales (CPU). Se muestran también los ajustes de tipo $t \propto N^a$ con $a \approx 2$ para ambos.

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, 25

[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] Keeling, M. J., Rohani, P. Modeling infectious diseases in humans and animals. págs. 15–26. Princeton University Press, 2008.

[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, 12, 17

[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] Barabási, A.-L., Stanley, H. E., *et al.* Fractal concepts in surface growth. Cambridge university press, 1995. 25

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.