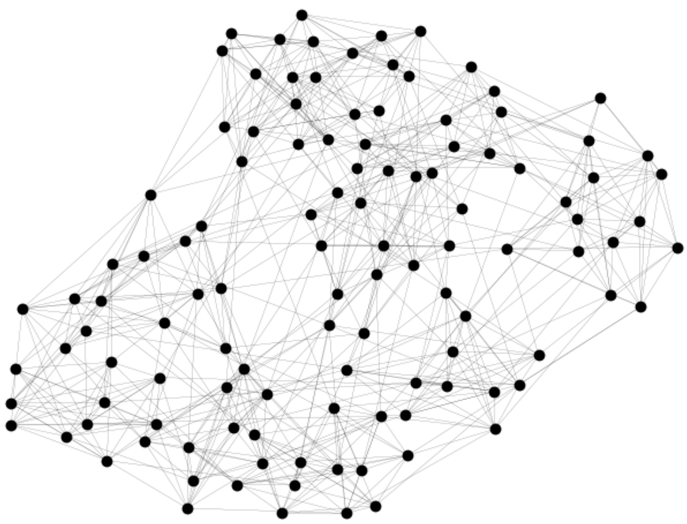
Universidad de Granada
Máster en Física y Matemáticas
Departamento de Física Estadística

Ejercicios de Redes complejas en fisica y neurociencia

Bartolomé Ortiz Viso



Mayo 2018

Abstract

Este trabajo compone la resolucion de algunos de los ejercicios propuestos en la asignatura del Master de Fisica y Matematicas fisymat de la Universidad de Granada denominada Fisica de redes complejas. La primera parte de la asignatura es impartida por el profesor Joaquín Torres.

Generalmente la mayoría de los ejercicios poseen algún componente computacional, ya sean cálculos, simulación o visualización. Con el fin de resaltar la importancia de este aspecto, pero no dificultar la lectura, los códigos están adjuntos y se pueden consultar en los apéndices dejando solo para el grueso del texto las conclusiones derivadas y las visualizaciones.

Así mismo muchos de los códigos empleados tambien poseen un salida visual en forma de animaciones y videos que puede ser interesante observar. Para este fin puede consultar el respositorio en github donde están todos los códigos utilizados, las salidas de los mismos y por supuesto, una versión de este documento en pdf y latex bajo la licencia creative commons:

`http://www.github.com/thebooort/Complex-Networks-and-Neuroscience`

Algunos de los códigos utilizados utilizan partes de otras librerías o códigos. En esos casos quedan citados los otros responsables y la licencia se adhiere a la que pusieran sus creadores iniciales antes de mis modificaciones para este trabajo.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Sistemas complejos

1.1. Atractores

Simular y analizar el comportamiento complejo del atrayente de Lorenz

El descubrimiento de este atractor data de 1963, cuando Edward Norton Lorenz desarrolló un modelo matemático simplificado para la convección atmosférica. El modelo es un sistema de ecuaciones diferenciales ordinarias compuesto por tres ecuaciones ahora conocidas como las ecuaciones de Lorenz:

$$\begin{cases} \frac{\mathrm{d}x}{\mathrm{d}t} & = \sigma(y - x), \\ \frac{\mathrm{d}y}{\mathrm{d}t} & = x(\rho - z) - y, \\ \frac{\mathrm{d}z}{\mathrm{d}t} & = xy - \beta z. \end{cases}$$

Normalmente se asume que los parametros σ, ρ, y β son positivos. El interés en este modelo proviene de que para una seleccion de valores como $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$ el sistema exhibe un comportamiento caótico.

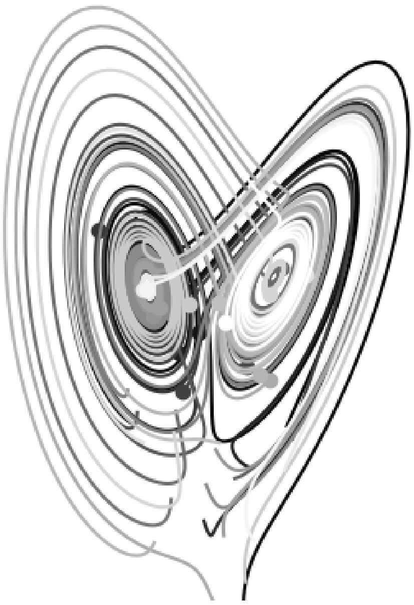


Figura 1.1: Atractor de Lorentz

Si $\rho < 1$ entonces sólo hay un punto de equilibrio en el origen. Este punto además es un atractor global.

Si $\rho = 1$ entonces la en nuestra dinámica aparece una bifurcacion de pitchfork , y para $\rho > 1$ apareceen dos puntos de equilibrio, a saber $\left(\sqrt{\beta(\rho-1)}, \sqrt{\beta(\rho-1)}, \rho-1\right)$ and $\left(-\sqrt{\beta(\rho-1)}, -\sqrt{\beta(\rho-1)}, \rho-1\right)$. Este par de puntos será estable si : $\rho < \sigma\frac{\sigma+\beta+3}{\sigma-\beta-1}$

Como avanzábamos para valores como $\rho = 28$, $\sigma = 10$, and $\beta = 8/3$, el sistema de Lorenz tiene soluciones caóticas (es decir que, aunque no todas las soluciones son caóticas algunas de ellas si pueden presentar este comportamiento).

Casi todos los puntos iniciales tenderán a un conjunto invariable que es lo que denominamos el atractor de Lorenz (véase figura:1.1), cuya forma geométrica es reconocible. Este comportamiento caótico se refleja en la gran sensibilidad del atractor frente a cambios en las condiciones iniciales como se puede ver en la grafica 1.2. *Las graficas pertenecen a extractos de animaciones que pueden consultarse en github.*

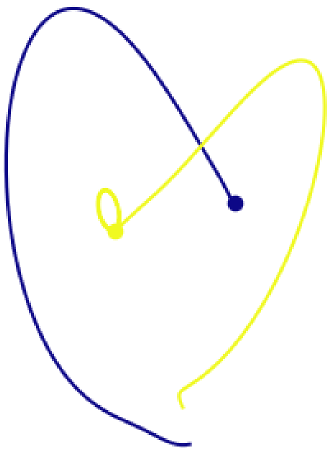


Figura 1.2: Atractor de Lorentz: variacion respecto a condiciones iniciales

Simular y analizar el comportamiento complejo del atrayente de Rossler

El atractor de Rössler es el atractor del sistema de Rössler, un sistema de tres ecuaciones diferenciales ordinarias no lineales estudiadas por Otto E. Rössler. Estas ecuaciones diferenciales definen un sistema dinámico del tiempo-continuo que muestra dinámicas caóticas asociadas con las propiedades fractales del atractor:

$$\begin{cases} \frac{dx}{dt} & = -y - z \\ \frac{dy}{dt} & = x + ay, \\ \frac{dz}{dt} & = b + z(x - c). \end{cases}$$

Rössler estudió el atractor caótico con $a = 0{,}2$, $b = 0{,}2$ y $c = 5{,}7$, aunque las propiedades de $a = 0{,}1$, $b = 0{,}1$, y $c = 14$ han sido más comúnmente utilizadas desde entonces. De nuevo con estos parámetros podemos encontrar orbitas con comportamiento caótico como se representa en la figura 1.3. *La grafica pertenecen a extractos de animaciones que pueden consultarse en github.*

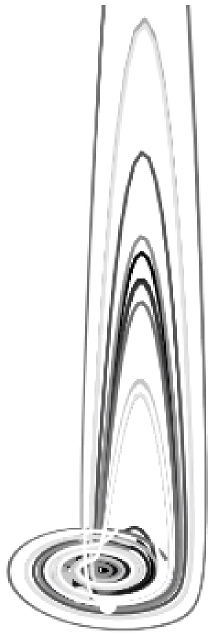


Figura 1.3: Atractor de Rossler

1.2. Juego de la vida

Simular el juego de la vida y estudiar su comportamiento emergente

El juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970.

Hizo su primera aparición pública en el número de octubre de 1970 de la revista Scientific American, en la columna de juegos matemáticos de Martin Gardner. Desde un punto de vista teórico, es interesante porque es equivalente a una máquina universal de Turing, es decir, todo lo que se puede computar algorítmicamente se puede computar en el juego de la vida. Para profundizar un poco más en el trabajo se ha desarrollado un programa para detectar si patrones implementados generan naves espaciales. Las naves espaciales son patrones que reaparecen en otra posición tras completar su período. Esto es, son patrones que tras un número finito de generaciones vuelven a su estado original pero en

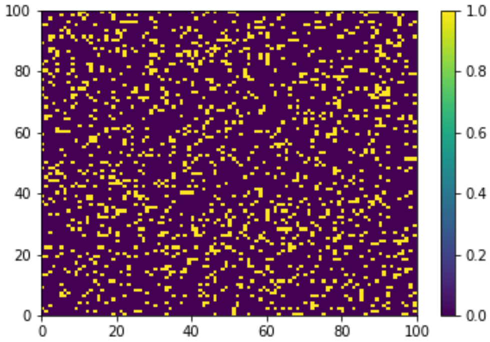


Figura 1.4: Disposicion inicial: partimos de una distribucion aleatoria de puntos en una malla

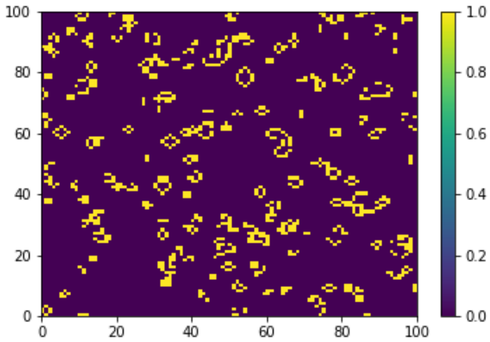


Figura 1.5: Evolucion del juego de la vida tras 5 pasos temporale

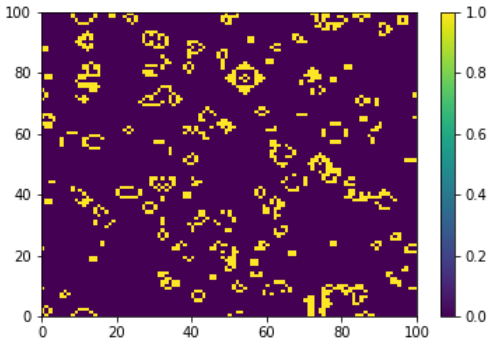


Figura 1.6: Evolucion del juego de la vida tras 10 pasos temporales

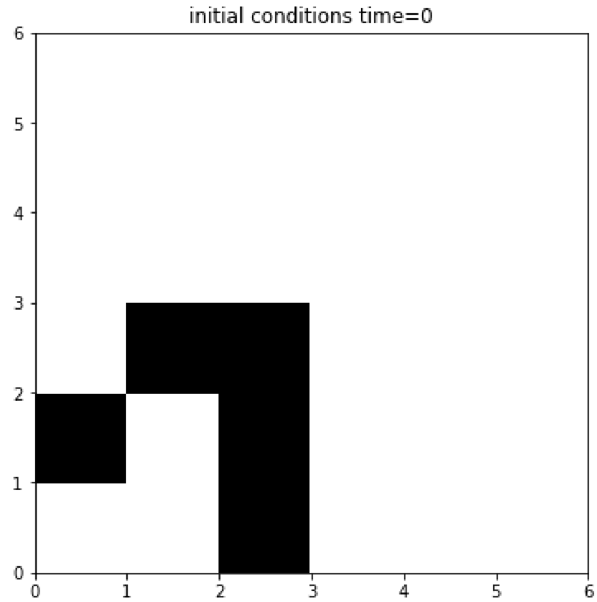


Figura 1.7: Disposicion inicial encontrada como buena

una ubicación diferente. En este caso se han utilizado las reglas estándar. Tras 4 pasos temporales se localizó que la figura era una nave espacial (figuras 1.7 y 1.8) Esta es la nave espacial más famosa, conocida como glider.

Modelo pilas de arena

El modelo de pilas de arena es un modelo matemático diseñado para analizar y explicar el comportamiento de autoorganización crítica a través de la teoría de grafos y la teoría de autómatas celulares utilizando herramientas algebraicas.

Si consideramos un plano con una gran cantidad de arena al que se le va añadiendo un poco cada paso temporal, si la pendiente es muy alta, la arena tenderá a caer provocando una avalancha, pues es un estado insetable. Esto se poducirá hasta que todo llegue a un estado estable en cuanto a la pendiente.

Se pueden ver el resultado de las simulaciones en 1.9,1.10,1.11

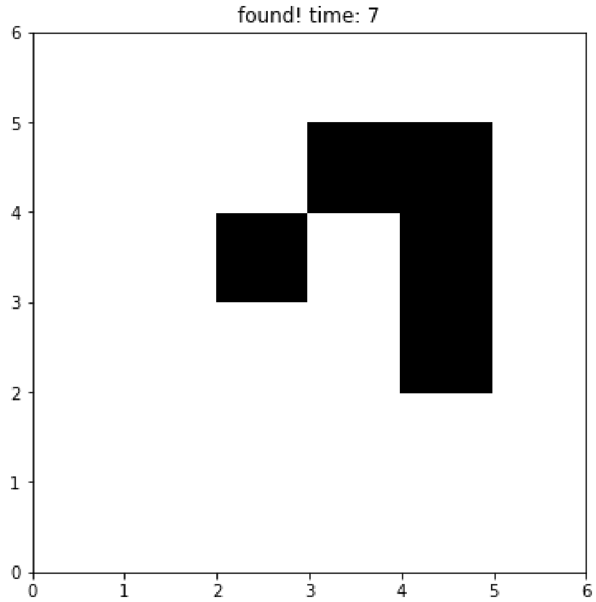


Figura 1.8: Identidad identica transportadad tras 4 pasos temporales

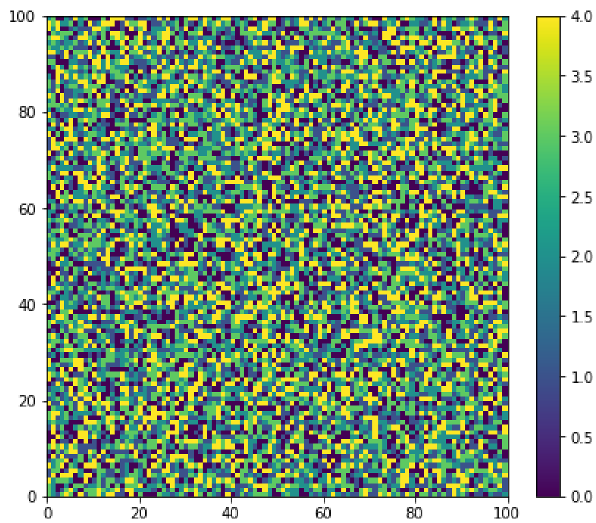


Figura 1.9: Pila de arena con condiciones iniciales aleatorias

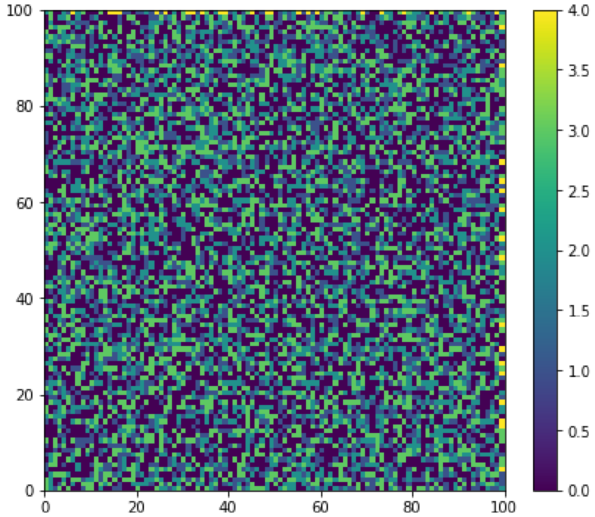


Figura 1.10: Sin adicion y partiendo de 1.9 vemos como el sistema tiene a producir avalanchas hasta nivelarse

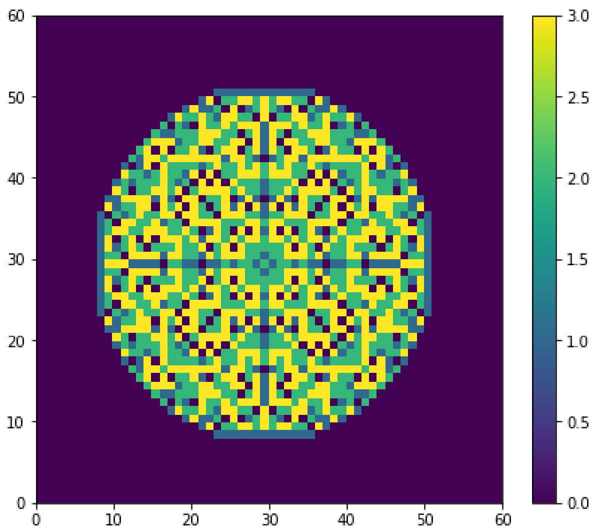


Figura 1.11: Creacion de fractales cuando añadimos un goteo de arena en el centro durante 100 pasos de tiempo

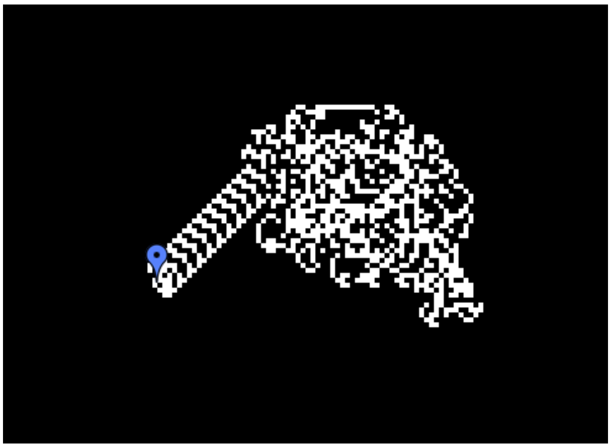


Figura 1.12: Hormiga tras 11000 pasos temporales. Las condiciones iniciales son nulas salvo la casilla de la hormiga

Modelo hormiga de langton

Aunque no partía como ejercicio, buscando información sobre algunas de turing y el juego de la vida también he encontrado otra sencilla maquina con comportamientos emergentes que dejo a continuación. La hormiga de Langton es un una máquina de Turing bidimensional con un conjunto de reglas muy sencillo, que sin embargo da lugar a comportamientos emergentes complejos. La hormiga de Langton clásica opera sobre una rejilla espacial cuadrada, en que cada celda puede estar en uno de dos estados (blanca o negra, 1 o 0, viva o muerta, etc). Fue inventada por Chris Langton en 1986 y su universalidad se demostró en el año 2000.

La hormiga siempre está mirando en una de las cuatro direcciones cardinales y se mueve un cuadrado cada vez, de acuerdo con las siguientes reglas:

- Si está sobre un cuadrado blanco, cambia el color del cuadrado, gira noventa grados a la izquierda y avanza un cuadrado.
- Si está sobre un cuadrado negro, cambia el color del cuadrado, gira noventa grados a la derecha y avanza un cuadrado.

1.3. Dimensiones de Haussdorf: variedades continuas y fractales

Demostrar que con la definición de dimension de Haussdorf se tiene (para variedades continuas:curva, superficie, y volumen) las dimensiones esperadas.

Partimos de la formula descrita en los apuntes, la dimension de Haussdorf (D) de una variedad viene determinada por:

$$D = \lim_{a \to 0} \left(\frac{-ln(N(a))}{ln(a)} \right) \tag{1.1}$$

donde $N(a)$ es el número mínimo de $esferas$ *d–dimensionales* necesarias para cubrir por completo la variedad en cuestión. Teniendo en cuenta que $dT \leq D \leq d$, donde d es la dinemsion euclídea y d_T e sla dimension topológica, sabemos de antemano que como todas nuestras variedades son continuas, el valor será $1, 2, 3$ para la curva, la superficie y le columen, respectivamente. Analizamos caso por caso:

- Curva continua:

  En este caso las $esferas$ *d–dimensionales* son segmentos de longitud $2a$, por tanto, sea una curva continua con longitud 1:

$$\begin{aligned} D &= \lim_{a \to 0} \left(\frac{-ln(N(a))}{ln(a)} \right) = -\lim_{a \to 0} \left(\frac{ln(1/2a)}{ln(a)} \right) \\ &= -\lim_{a \to 0} \left(\frac{\frac{d}{da} ln(1/2a)}{\frac{d}{da} ln(a)} \right) = -\lim_{a \to 0} \left(\frac{-1/x}{1/x} \right) = 1 \end{aligned} \tag{1.2}$$

- Superficie continua:

  En este caso las $esferas$ *d–dimensionales* son discos de radio a, por tanto, dada una superficie, podemos recubir su borde colocando esferas que abarcan una longitud de

$2a$ a lo alto y a lo largo:

$$
\begin{aligned}
D = \lim_{a\to 0}(\frac{-ln(N(a))}{ln(a)}) &= -\lim_{a\to 0}(\frac{ln(1/(2a)^2)}{ln(a)}) \\
= -\lim_{a\to 0}(\frac{\frac{d}{da}ln(1/4x^2)}{\frac{d}{da}ln(a)}) &= -\lim_{a\to 0}(\frac{-2/x}{1/x}) = 2
\end{aligned}
\tag{1.3}
$$

- Volumen continuo:

  En este caso las *esferas d–dimensionales* son esferas de radio $2a$, por tanto, dado un volumen continuo podemos recubir su borde colocando esferas que abarcan una longitud de $2a$ a lo alto, a lo ancho y a lo largo :

$$
\begin{aligned}
D = \lim_{a\to 0}(\frac{-ln(N(a))}{ln(a)}) &= -\lim_{a\to 0}(\frac{ln(1/(2a)^3)}{ln(a)}) \\
= -\lim_{a\to 0}(\frac{\frac{d}{da}ln(1/8x^3)}{\frac{d}{da}ln(a)}) &= -\lim_{a\to 0}(\frac{-3/x}{1/x}) = 3
\end{aligned}
\tag{1.4}
$$

Calcular la dimensión Hausdorff de los siguientes fractales: Conjunto de Cantor, la Isla de Koch, la Alfombra de Sierpinsky y la triangulo de Sierpinsky

Atendiendo a la definición de las notas, tenemos que

$$M \sim R^D$$

donde D es la dimensión fractal. Por tanto si desarrollamos obtenemos:

$$D = \frac{log(M)}{log(R)}$$

Pasemos a analizar cada uno de los casos.

- Conjunto de cantor:

en este caso tenemos una variedad de masa 1, de manera que en cada paso obtenemos dos subvariedades, estas subvariedades coinciden con la original si aumentamos la escala 3 unidades de la que tenemos, por tanto:

$$D = \frac{log(2)}{log(3)}$$

- Isla de Koch:

  en este caso tenemos una variedad cuya dimension D debe ser $1 < D < 2$. En esta figura si aplicamos un factor de multiplicacion 3 obtenemos 4 veces la seccion inicial que habíamos aumentado, por tanto:

$$D = \frac{log(4)}{log(3)}$$

- Alfombra de Sierpinsky:

  en este caso tenemos una variedad cuya dimension D debe ser $1 < D < 2$. Si aumentamos la figura por un factor 3, obtenemos 8 subvariedades identicas a la original , por tanto:

$$D = \frac{log(8)}{log(3)}$$

- triangulo de Sierpinsky:

  en este caso tenemos una variedad cuya dimension D debe ser $1 < D < 2$. Si aumentamos la figura por un factor 2, obtenemos 3 subvariedades identicas a la original , por tanto:

$$D = \frac{log(3)}{log(2)}$$

Capítulo 2

Redes complejas

2.1. Grafos completos

Dibuja L_i, $i = 1, ..., 8$

A continuación se muestran todos los grafos generados a partir del script del anexo.

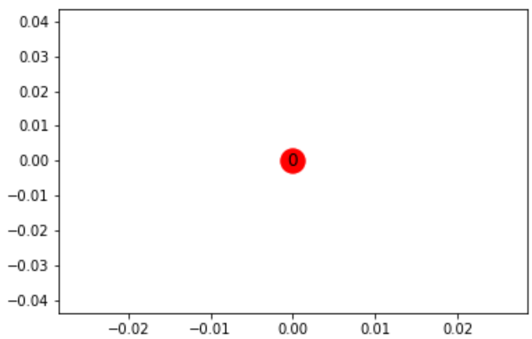


Figura 2.1: Grafo L_1

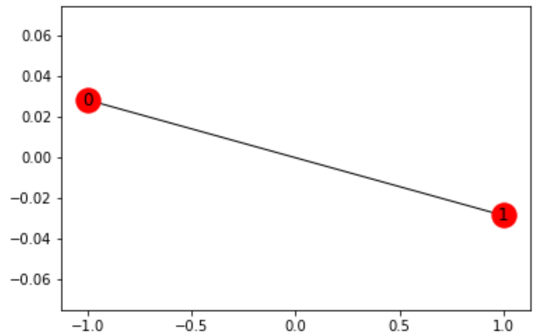


Figura 2.2: Grafo L_2

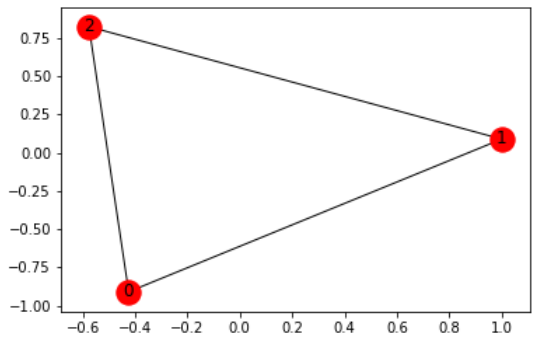


Figura 2.3: Grafo L_3

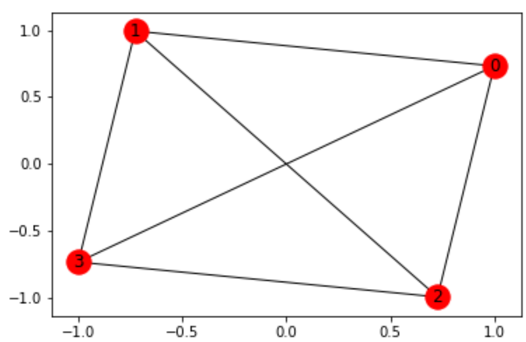


Figura 2.4: Grafo L_4

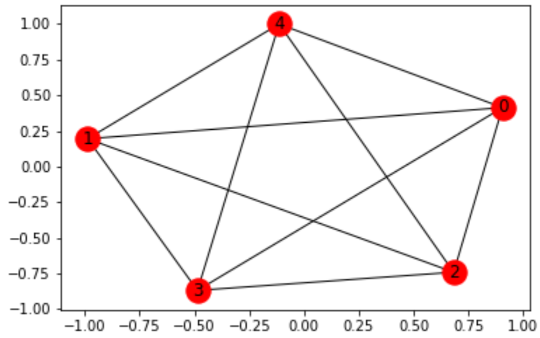


Figura 2.5: Grafo L_5

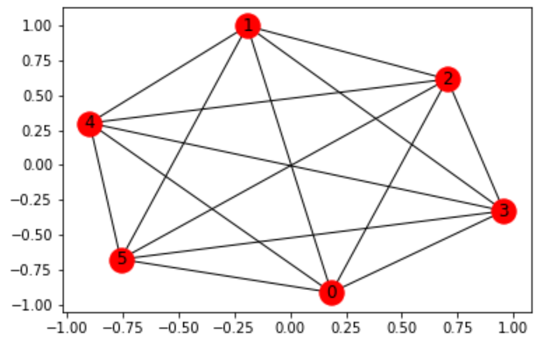


Figura 2.6: Grafo L_6

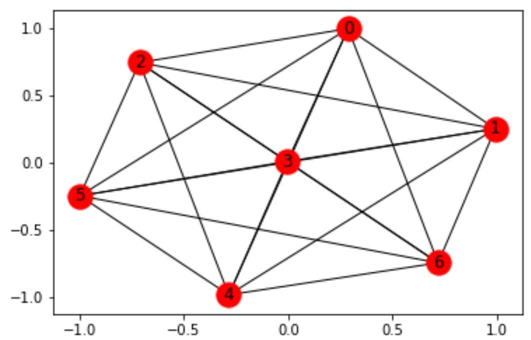


Figura 2.7: Grafo L_7

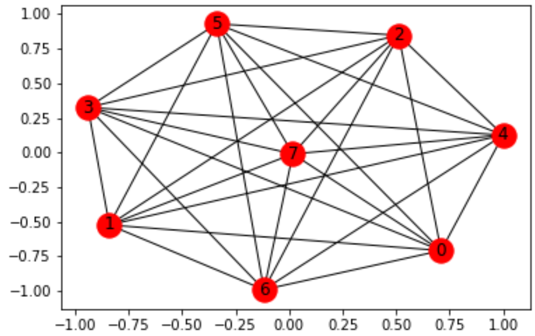


Figura 2.8: Grafo L_8

2.2. Multigrafo

Calcular la lista de aristas y la matriz de adyacencia para el multigrafo representado en la figura 5

Lista de aristas

(n_1, n_2)
(n_2, n_1)
(n_2, n_3)
(n_2, n_4)
(n_4, n_4)
(n_4, n_3)
(n_3, n_5)
(n_5, n_4)

Matriz de Adyacencia:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

2.3. Grafo pesado

Calcular la lista de aristas y la matriz de adyacencia del grafo pesado que aparece en la figura 5

Lista de aristas

$(n_1, n_2, 5)$
$(n_2, n_3, 1)$
$(n_2, n_4, 6)$
$(n_3, n_5, 5)$
$(n_4, n_3, 3)$
$(n_5, n_4, 1)$

Matriz de Adyacencia:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 6 & 0 & 3 & 1 \\ 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

2.4. Grafo bipartito

Calcular la matriz de incidencia del grafo bipartito mostrado en el figura 6. Cual sería la matriz de incidencia si no existiera en dicho grafo el nodo m5 ?

Matriz de Incidencia:

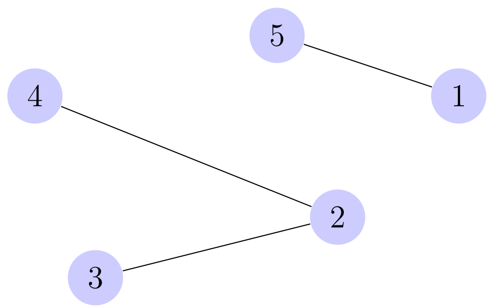
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Matriz de Incidencia sin nodo m_5:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

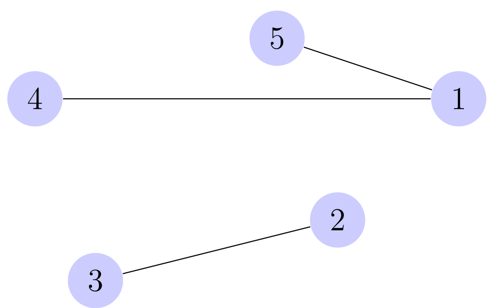
2.5. Grafo proyección 1

Dibujar el grafo proyección de nodos del grafo bipartito de la figura 6 y demostrar que el grafo proyección de nodos viene descrito por una matriz de adyacencia P tal que $P = BB^T$



$$P = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Dibujar el grafo proyección de grupos del grafo bipartito de la figura 6 y demostrar que el grafo proyección de grupos viene descrito por una matriz de adyacencia P tal que $P = BB^T$



$$P = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

En un grafo simple no dirigido $k_i \in [0, N-1]$, y en un red dirigida $k_i^{in} \in [0, N-1]$ y $k_i^{out} \in [0, N-1]$.

Ejercicio propuesto en clase

El resultado es evidente, puesto que en un grafo no dirigido como maximo el nodo puede estar conectado con todos los otros nodos, menos él mismo, lo cual hace que esté en un rango entre 0 (si no esta conectado a nadie) y N-1 cuando esta conectado a todos los otros nodos. En los otros casos, el razonamiento es el mismo puesto que se habla de grafos dirigidos y no multigrafos (en los cuales se llegaría hasta N)

Se tiene que para una red simple $L = \frac{1}{2}\langle k \rangle N$

Ejercicio propuesto en clase

Tenemos por definición que:

$$L = \frac{1}{2}\langle k \rangle N = \frac{1}{2}\frac{1}{N}(\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij})N = \frac{1}{2}(\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij})$$

donde la ultima igualdad es trivial puesto que en la matriz de adyacencia aparece el numero de aristas repetido dos veces.

Calcular la distribución de probabilidad de grados del grafo no dirigido de la figura 5 y ver que está normalizada

- $P(1) = 1/5$
- $P(2) = 1/5$
- $P(3) = 3/5$

$P(1) + P(2) + P(3) = 1$

Calcular la distribución de probabilidad de grados del grafo dirigido y del multigrafo de la figura 5 y ver que está normalizada

Grafo dirigido:

Saliente

- $P(1) = 4/5$
- $P(2) = 1/5$

$P(1) + P(2) = 1$

Entrante

- $P(0) = 1/5$
- $P(1) = 2/5$
- $P(2) = 2/5$

$P(1) + P(2) + P(0) = 1$

Multigrafo:

Saliente

- $P(1) = 3/5$
- $P(2) = 1/5$
- $P(3) = 1/5$

$P(1) + P(2) + P(3) = 1$

Entrante

- $P(1) = 3/5$
- $P(2) = 1/5$
- $P(3) = 1/5$

$P(1) + P(2) + P(3) = 1$

En el grafo no dirigido de la figura 5 calcular el número total de caminos cíclicos de longitud n = 3 que empiezan en los nodos

$$N = Traza\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}^3 = Traza\begin{pmatrix} 0 & 3 & 1 & 1 & 2 \\ 3 & 2 & 6 & 6 & 2 \\ 1 & 6 & 4 & 5 & 5 \\ 1 & 6 & 5 & 4 & 5 \\ 2 & 2 & 5 & 5 & 2 \end{pmatrix} = 12$$

Si el número nos parece grande, debemos recordar que no hablamos de caminos cíclicos auto evitados.

Demostrar que para una red regular en forma de anillo donde cada nodo tiene grado k = 2m el coeficiente de agrupamiento es $C = 3(m-1)/2(2m-1)$

Sea un grafo con grado $K = 2m$

Para calcularlo el coeficiente de agrupamiento solo tenemos que tener en cuenta el numero de triángulos en el grafo de manera que uno de los nodos del triangulo sea nuestro nodo y el numero de tripletas en el grafo.

El numero de tripletas es el mismo que el de posibles links entre nodos vecinos de i, con lo que lo unico que tenemos que calcular es el orden del grafo completo que tuviese "m"links:

$$N_t ripletas = \frac{2m(2m-1)}{2} = m(2m-1)$$

Para el numero de triángulos basta ver:

$$N_{triangulos} = 3\sum_{j=0}^{m-1} j = 3\frac{m(m-1)}{2}$$

Con lo cual, dividiendo por la formula que tenemos en los apuntes:

$$C = \frac{1}{N}\sum_{i=1}^{N} \frac{3(m-1)}{2(2m-1)} = \frac{3(m-1)}{2(2m-1)}$$

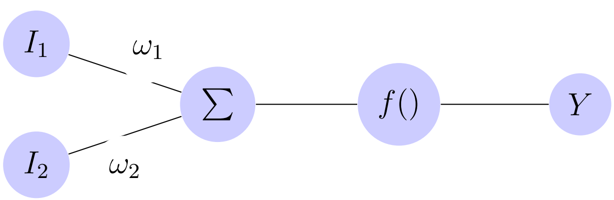
Capítulo 3

Redes de neuronas

3.1. Puertas logicas

Elaborar las puertas lógicas anteriores mediante una red neuronal de tipo McCulloch-Pitts eligiendo convenientemente los valores de los pesos y de los umbrales

Puerta lógica OR:

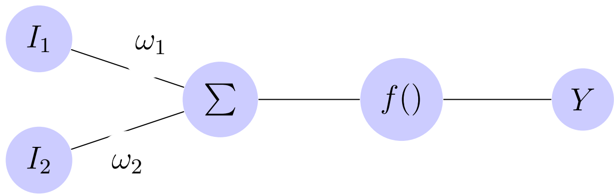


Parametros: $\omega_1 = 1$ y $\omega_2 = 1$

Funcion:

$$y = \begin{cases} 1, & \text{si} \quad y \geq 1 \\ 0, & \text{si} \quad y < 1 \end{cases}$$

Puerta lógica AND:

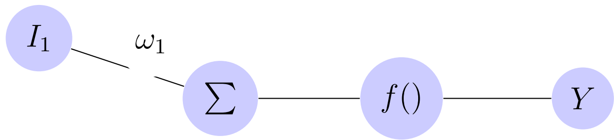


Parametros: $\omega_1 = 1$ y $\omega_2 = 1$

Funcion:

$$y = \begin{cases} 1, & \text{si} \quad y \geq 2 \\ 0, & \text{si} \quad y < 2 \end{cases}$$

Puerta lógica NOT:

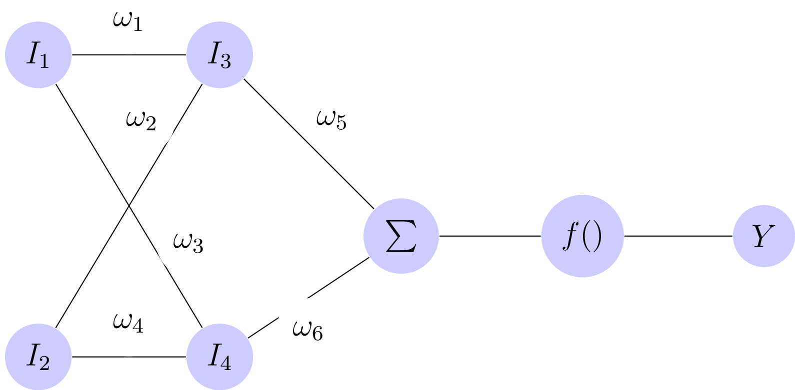


Parametros: $\omega_1 = 1$.

Funcion:

$$y = \begin{cases} 0, & \text{si} \quad y \geq 1 \\ 1, & \text{si} \quad y < 1 \end{cases}$$

Puerta lógica XOR:



Parametros: $\omega_1 = 2$,$\omega_2 = 2$,$\omega_3 = -1$,$\omega_4 = -1$,$\omega_6 = 2$ y $\omega_5 = 2$

Funcion:

$$y = \begin{cases} 1, & \text{si} \quad y \geq 1 \\ 0, & \text{si} \quad y < 1 \end{cases}$$

3.2. Hodgkin-Huxley

Simular el modelo de Hodgkin-Huxley y determinar el rango de I en el que aparecen oscilaciones

El modelo de Hodgkin y Huxley describe cómo se inician y transmiten los potenciales de acción en las neuronas. Consiste en un conjunto de ecuaciones diferenciales ordinarias no lineales que aproxima las características eléctricas de células excitables, en nuestro caso, las neuronas.

$$I = C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V_m - V_K) + \bar{g}_{\mathrm{Na}} m^3 h (V_m - V_{Na}) + \bar{g}_l (V_m - V_l),$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h$$

donde "I" es la corriente por unidad de area, y α_i and β_i son las ratios de los canales de iones que no depende del tiempo. $\bar{g}_n$ es el valor máximo de la conductancia. " n ", " m "

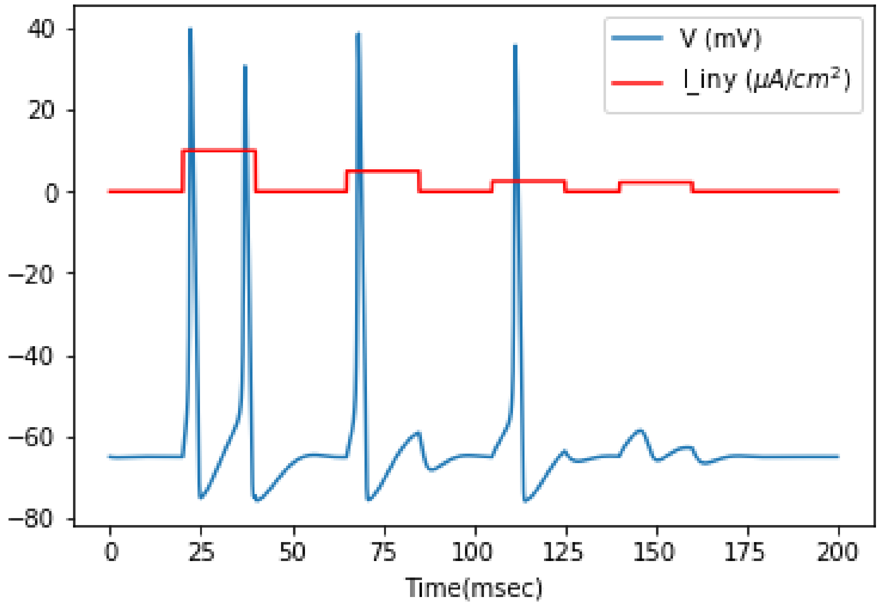


Figura 3.1: Representacion del modelo HH con la corriente

y " h " son cantidades adimensionales entre 0 y 1 que están asociadas con la activación del canal de potasio, la activación del canal de sodio y la inactivación del canal de sodio, respectivamente.

He concentrado en un grafico 3.1 la resolucion del ejercicio, de manera que se puede observar los rangos de I que producen actividad con un grafico superpuesto.

3.3. Modelo FN

Integrar numéricamente el modelo de FN y estudiar comportamientos dinámicos que pueden aparecer

El modelo de FitzHugh-Nagumo (FHN) describe un prototipo de un sistema excitable (por ejemplo, una neurona). Toma su nombre de Richard FitzHugh (1922 - 2007), quien propuso el modelo teórico en 1961, así como de J. Nagumo y otros, que construyeron un circuito electrónico equivalente.

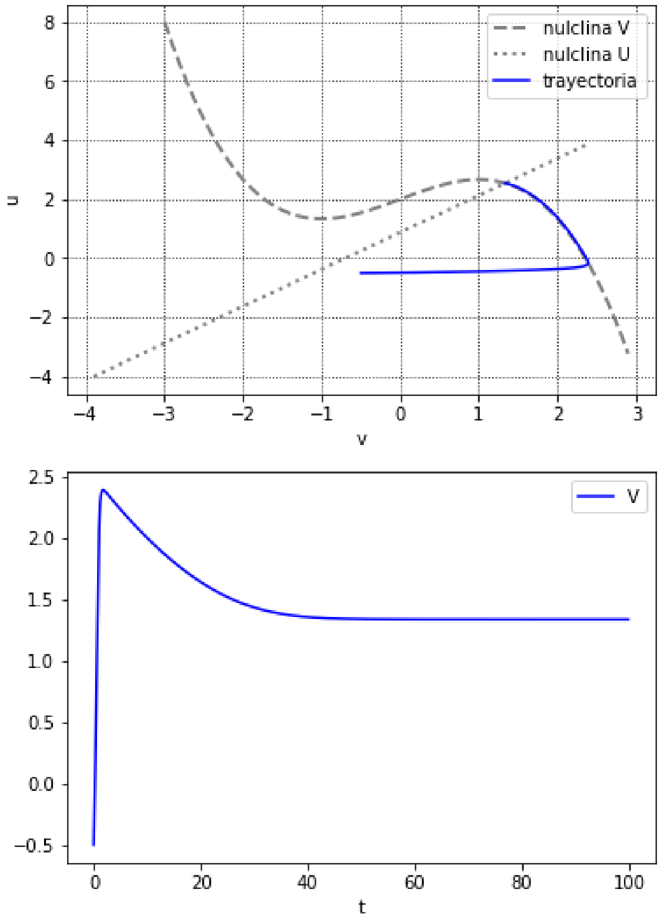


Figura 3.2: Tendencia al punto estable

$$\dot{v} = v - \frac{v^3}{3} - u + I_{\text{ext}}$$

$$\tau\dot{u} = v + a - bu$$

Para mostrar los distintos tipos de comportamientos se han creado una serie de graficas que los exhiben: 3.23.33.43.5.

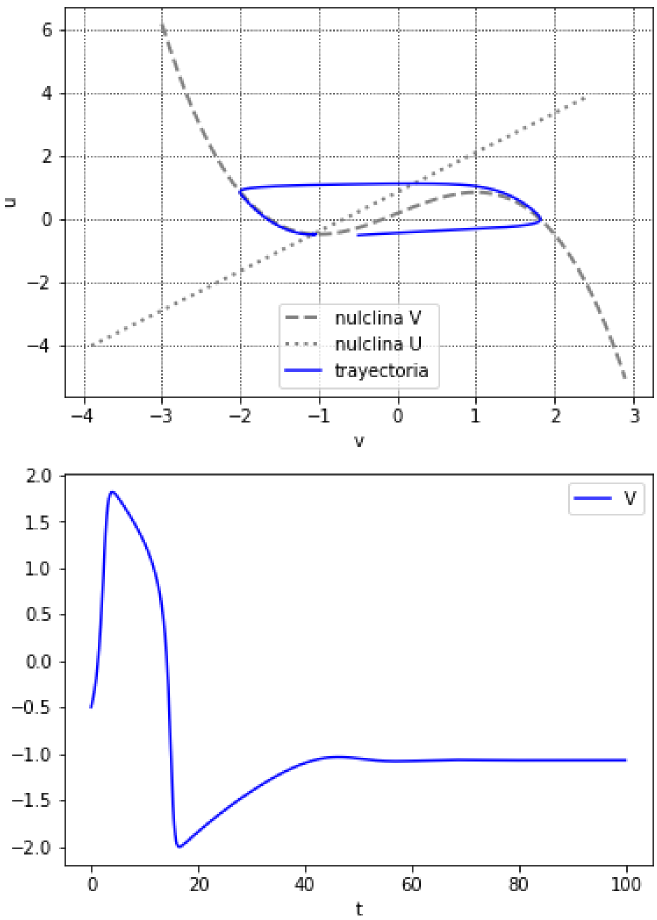


Figura 3.3: Tendencia al punto estable

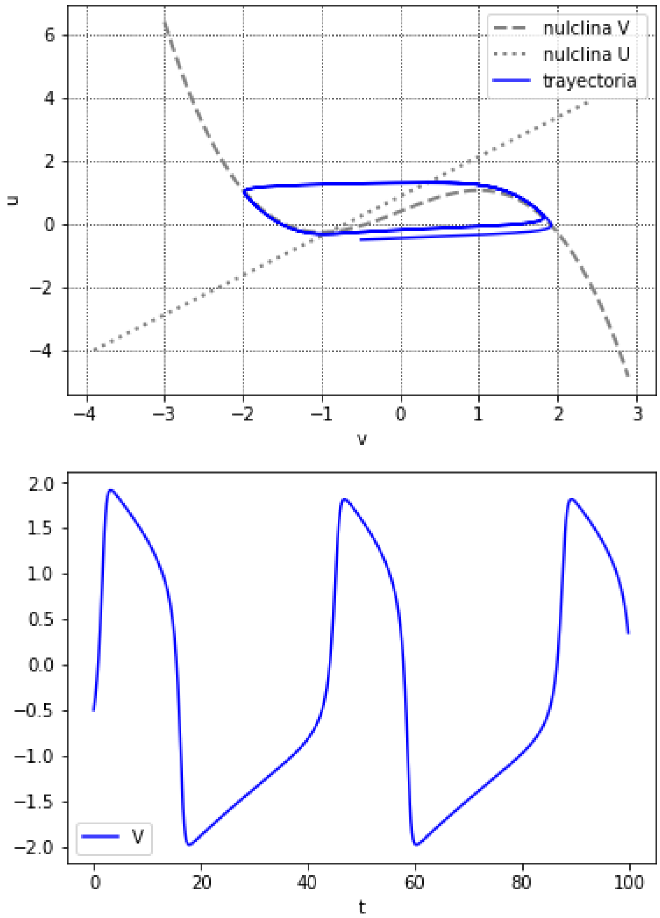


Figura 3.4: Si recibe suficiente fuerza, empiezan comportamientos periodicos

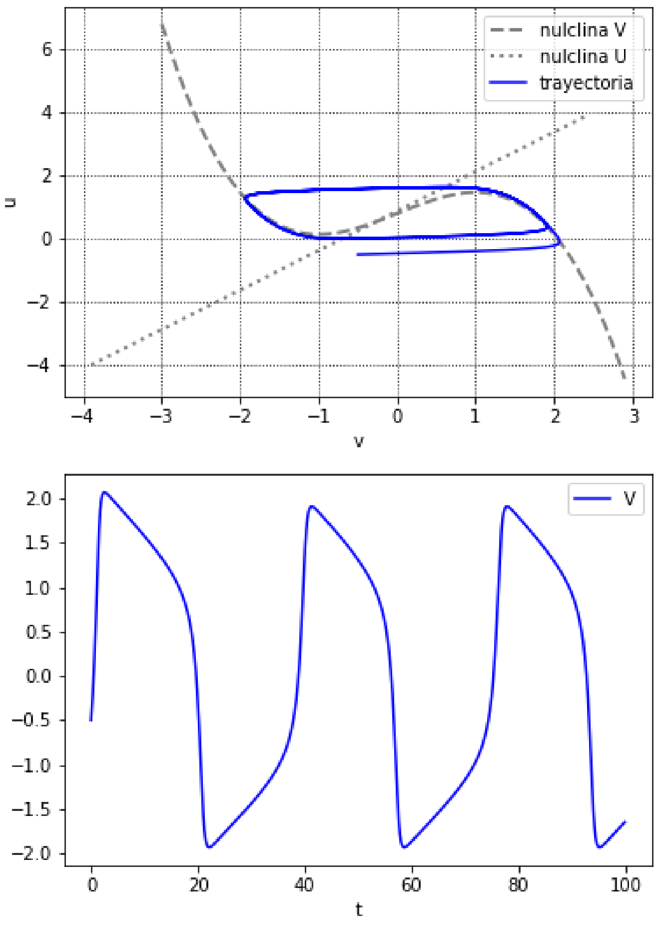


Figura 3.5: Comportamiento periodico

3.4. Modelo Tsodyks-Markram

Simular el modelo de Tsodyks-Markram de sinapsis dinámica y ver la forma del potencial postsináptico excitador EPSP usando por ejemplo un modelo lineal de integración y disparo para la neurona postsináptica cuando está sometida a un tren de pulsos presinápticos periódicos que llegan a una frecuencia f dada

Se puede consultar el script del intento en el apéndice 3, sin embargo, no ha dado resultados creíbles.

Capítulo 4

Redes sociales

4.1. Modelo del votante Barabasi-Albert y S-M

Simular el modelo del votante en una red pequeño mundo y en una red de Barabási-Albert y estudiar su comportamiento emergente

El modelo de votante es un modelo de no equilibrio que tiene solucion analitica en cualquier dimension. Tendriamos N agentes sociales, nodos de una red. Cada agente puede estar en dos estados (± 1). Cada paso temporal es elegido un agente y uno de sus vecinos y adquieren el mismo estado.

Red de pequeño mundo:

En la figura se puede ver el aspecto de nuestra red 4.1 (se ha creado segun los estándares, pero el output de python es bastante reducido). El aspecto final 4.2. Y la fraccion a lo largo del tiempo, propia de estos sistemas complejos: 4.3.

Red de Barabási-Albert:

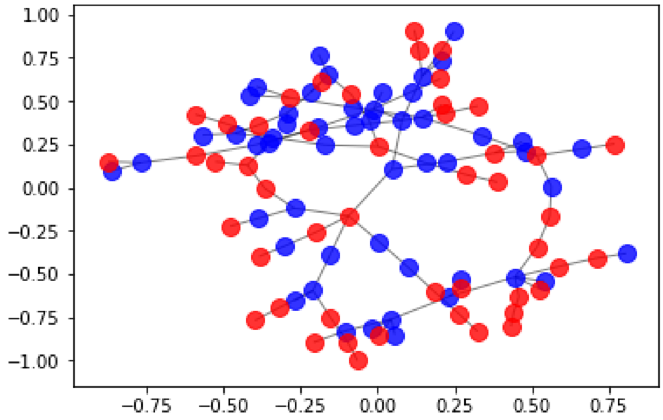


Figura 4.1: Estado inicial para red pequeño mundo

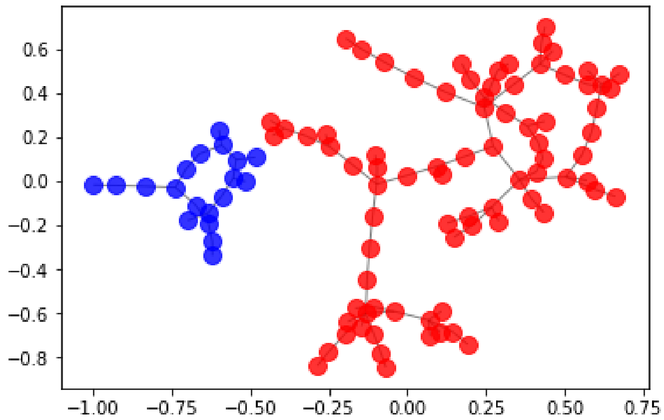


Figura 4.2: Estado final (Se representa distinto para observar mejor la clusterizacion)

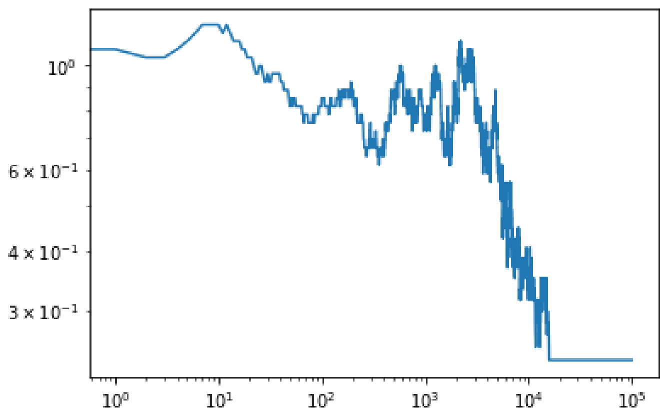


Figura 4.3: Fracción de pares con distinta opinión en función del tiempo en un modelo del votante

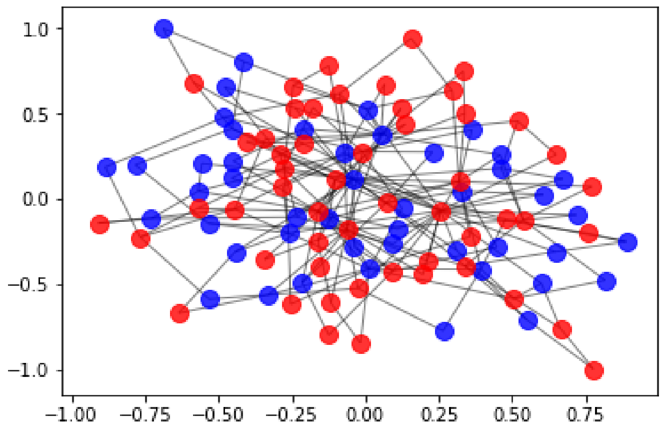


Figura 4.4: Estado inicial para red pequeño mundo

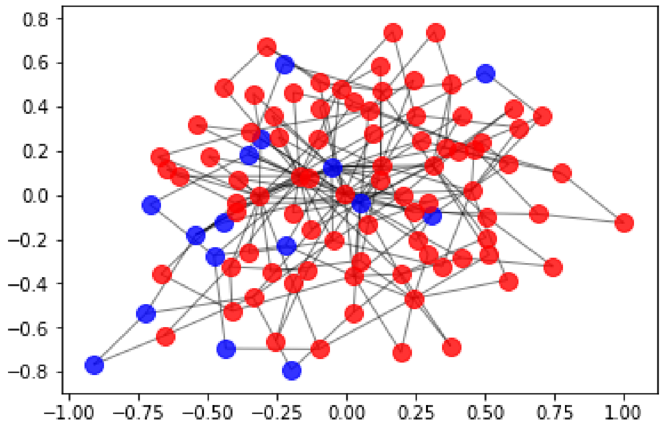


Figura 4.5: Estado final

En la figura se puede ver el aspecto de nuestra red 4.4 (se ha creado segun los estándares, pero el output de python es bastante reducido). El aspecto final4.5. Y la fraccion a lo largo del tiempo, propia de estos sistemas complejos: 4.6. En este caso vemos algunas incoherencias, pero son comprensibles debido al bajo numero de nodos de nuestra red (por limitaciones de mi ordenador).

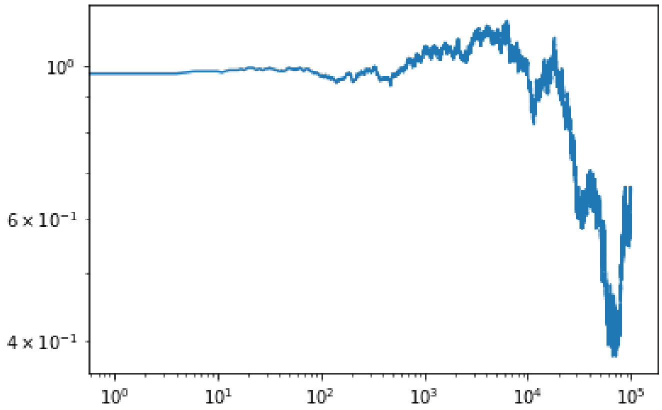


Figura 4.6: Fracción de pares con distinta opinión en función del tiempo en un modelo del votante

4.2. Competicion de lenguas

Estudiar las soluciones del sistema anterior y su estabilidad e interpretar cada uno de los estados estacionarios posibles.

Estamos ante el estudio del sistema de competición de lenguas, que viene dictaminado por la ecuación:

$$dm_A/dt = s(m_A)^a(1-m_A) - (1-s)(1-m_A)^a m_A$$

Esta ecuacion tiene 3 puntos de equilibrio para $a \neq 1$: Si $a > 1$ tenemos $m_A = 0$ y $m_A = 1$ como puntos de equilibro estables, aquellos en los que alguna otra lengua elimina a la competencia. Tenemos otro punto critico entre 0 y 1 que representa un estado inestable de equilibrio entre las lenguas.

Para $a < 1$ el problema es el mismo pero la cualidad de estabilidad se invierte en los puntos criticos encontrados. (Se ha consultado para este apartado el articulo *Microscopic Abrams-Strogatz model of language competition*)

Appendix A

Tema 1: Códigos

A.1. Atractores de Lorentz y Rossler

Atractor de Lorentz

```
# -*- coding: utf-8 -*-

"""
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Make 3D Animations of Lorentz Attractor with Matplotlib
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
code adapted from
"https://jakevdp.github.io/blog/2013/02/16/animating-the-lorentz-system-in
   -3d/"
and extracted from PYwonderland library.
This one can be found online in github
"""

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D

```

```python

# number of particles.
num_particles = 25

# constants for Lorenz system.
alpha = 10.0
beta = 8/3.0
gamma = 28.0


def derivative(point, t):
    """return the tangent direction at (x,y,z)."""
    x, y, z = point
    return [alpha * (y - x),
            x * (gamma - z) - y,
            x * y - beta * z]


fig = plt.figure(figsize=(6.4, 4.8), dpi=100)
ax = fig.add_axes([0, 0, 1, 1], projection='3d', xlim=(-25, 25),
                  ylim=(-35, 35), zlim=(5, 55), aspect=1)
ax.view_init(30, 0)
ax.axis('off')

lines = []
points = []
colors = plt.cm.gray(np.linspace(0, 1, num_particles))

for c in colors:
    lines.extend(ax.plot([], [], '-', c=c))
    points.extend(ax.plot([], [], 'o', c=c))


x0 = -15 + 30 * np.random.random((num_particles, 3))
t = np.linspace(0, 4, 1001)
x_t = np.array([odeint(derivative, point, t) for point in x0])


```

```
def init():
    for line, point in zip(lines, points):
        line.set_data([], [])
        line.set_3d_properties([])

        point.set_data([], [])
        point.set_3d_properties([])
    return lines + points


def animate(i):
    i = 2*i % x_t.shape[1]  # accelarate the animation.

    for line, point, x_j in zip(lines, points, x_t):
        x, y, z = x_j[:i].T
        line.set_data(x, y)
        line.set_3d_properties(z)

        # note that plot() receives a list parameter so we have
        # to write x[-1:] instead of x[-1]!
        point.set_data(x[-1:], y[-1:])
        point.set_3d_properties(z[-1:])

    ax.view_init(30, 0.3*i)
    fig.canvas.draw()
    return lines + points


anim = FuncAnimation(fig, animate, init_func=init, interval=5,
                     frames=500, blit=True)

anim.save('lorenz.mp4', writer='ffmpeg', fps=30, dpi=200,
          codec='libx264', extra_args=['-crf', '20'])
```

Atractor de Rossler

```
# -*- coding: utf-8 -*-
"""
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
Make 3D Animations of Lorenz Attractor with Matplotlib
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

code adapted from
"https://jakevdp.github.io/blog/2013/02/16/animating-the-lorentz-system-in
    -3d/"
and modified also from PYwonderland library.
This one cannot be found online because it is a modification I made for
    this document, you can find it at my github

"""

import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D


# number of particles.
num_particles = 20

# constants for Lorenz system.
alpha = 0.2
beta = 0.2
gamma = 8.0


def derivative(point, t):
    """return the tangent direction at (x,y,z)."""
    x, y, z = point
    return [-y - z,
            x + alpha * y,
            beta + z * ( x - gamma )]


fig = plt.figure(figsize=(9.1, 7.2), dpi=100)
ax = fig.add_axes([0, 0, 1, 1], projection='3d', xlim=(-70, 70),
```

```
                  ylim=(-70, 70), zlim=(-10, 70), aspect=1)
ax.view_init(30, 0)
ax.axis('off')

lines = []
points = []
colors = plt.cm.gist_ncar(np.linspace(0, 1, num_particles))

for c in colors:
    lines.extend(ax.plot([], [], '-', c=c))
    points.extend(ax.plot([], [], 'o', c=c))


x0 =  -5 + 20 * np.random.random((num_particles, 3))
for i, element in enumerate(x0):
    if element[2] != 2:
        element[2] = 0.0
t = np.linspace(0, 80, 3001)
x_t = np.array([odeint(derivative, point, t) for point in x0])


def init():
    for line, point in zip(lines, points):
        line.set_data([], [])
        line.set_3d_properties([])

        point.set_data([], [])
        point.set_3d_properties([])
    return lines + points


def animate(i):
    i = 2*i % x_t.shape[1]  # accelarate the animation.

    for line, point, x_j in zip(lines, points, x_t):
        x, y, z = x_j[:i].T
        line.set_data(x, y)
        line.set_3d_properties(z)
```

```python

        # note that plot() receives a list parameter so we have
        # to write x[-1:] instead of x[-1]!
        point.set_data(x[-1:], y[-1:])
        point.set_3d_properties(z[-1:])

    ax.view_init(30, 0.3*i)
    fig.canvas.draw()
    return lines + points


anim = FuncAnimation(fig, animate, init_func=init, interval=5,
                     frames=500, blit=True)

anim.save('rossler.mp4', writer='ffmpeg', fps=30, dpi=200,
          codec='libx264', extra_args=['-crf', '20'])
```

A.2. Autómatas celulares

Juego de la vida

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  5 18:30:57 2018

@author: booort & stackoverflow answer
"""

import numpy
import matplotlib.pyplot as plt

# rules
def play_life(a):
    xmax, ymax = a.shape
    b = a.copy() # copy grid & Rule 2
    for x in range(xmax):
```

```
        for y in range(ymax):
            n = numpy.sum(a[max(x - 1, 0):min(x + 2, xmax), max(y - 1, 0):
    min(y + 2, ymax)]) - a[x, y]
            if a[x, y]:
                if n < 2 or n > 3:
                    b[x, y] = 0 # Rule 1 and 3
            elif n == 3:
                b[x, y] = 1 # Rule 4
    return(b)

# grid start
life = numpy.random.random_integers(0,1,(100,100))
plt.figure(figsize=(6, 6))
plt.pcolormesh(life)
plt.show
# evolution
for i in range(104):
    life = play_life(life)
plt.figure(figsize=(6, 6))
plt.pcolormesh(life)
plt.show
```

Buscador para el juego de la vida

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  5 18:42:40 2018

@author: booort
"""

#game of life, life searcher


import numpy
import matplotlib.pyplot as plt

# rules
```

```
def play_life(a):
    xmax, ymax = a.shape
    b = a.copy() # copy grid & Rule 2
    for x in range(xmax):
        for y in range(ymax):
            n = numpy.sum(a[max(x - 1, 0):min(x + 2, xmax), max(y - 1, 0):
   min(y + 2, ymax)]) - a[x, y]
            if a[x, y]:
                if n < 2 or n > 3:
                    b[x, y] = 0 # Rule 1 and 3
            elif n == 3:
                b[x, y] = 1 # Rule 4
    return(b)

# space ship searcher: compares 3x3 grid searching for the same pattern
def check_life(a,b):
    xmax, ymax =a.shape
    result=False
    for x in range(xmax-2):
        for y in range(ymax-2):
                sample_1=a[numpy.ix_([x,x+1,x+2],[y,y+1,y+2])]
                sample_2=(sample_1==b).all()
                if sample_2==True:
                    result=True
    return result

# initial conditions
experiment=numpy.zeros((6,6))
experiment[0,2]=1
experiment[1,2]=1
experiment[1,0]=1
experiment[2,2]=1
experiment[2,1]=1

plt.figure(figsize=(6, 6))
plt.title('initial conditions time={}'.format(0))
plt.pcolormesh(experiment,cmap='binary')
plt.show
```

```

# our spaceship that is hopefully alive
space_ship=numpy.zeros((3,3))
space_ship[0,2]=1
space_ship[1,2]=1
space_ship[1,0]=1
space_ship[2,2]=1
space_ship[2,1]=1

plt.figure(figsize=(6, 6))
plt.title('spaceship')
plt.pcolormesh(space_ship,cmap='binary')
plt.show

# temporal evolution + searching

for i in range(100):
    experiment = play_life(experiment)
    detection = check_life(experiment,space_ship)
    print(detection)
    if detection==True:
        plt.figure(figsize=(6, 6))
        plt.title('found! time: {}' .format(i))
        plt.pcolormesh(experiment,cmap='binary')
        plt.show

plt.figure(figsize=(6, 6))
plt.title('finals conditions')
plt.pcolormesh(experiment,cmap='binary')
plt.show
```

Pila de arena

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  5 17:55:48 2018

@author: booort
```

```
"""

import numpy
import matplotlib.pyplot as plt

# sandpile model
def play_sand(a):
    xmax, ymax = a.shape
    b = a.copy() # copy grid
    for x in range(xmax-2):
        for y in range(ymax-2):
            if a[x, y]>=4:
                b[x,y]=0
                if (x>0 and y>0):
                    b[x+1,y] +=1
                    b[x-1,y] +=1
                    b[x,y+1] +=1
                    b[x,y-1] +=1
                """ # uncomment for additional features
                    b[x+1,y+1] +=1
                    b[x+1,y-1] +=0
                    b[x-1,y+1] +=0
                    b[x-1,y-1] +=1"""

    return(b)

# un comment to see random configuration

# sand = numpy.random.random_integers(0,4,(100,100))

# example of cascade
sand = numpy.zeros((100,100))
sand[49,49]=3
plt.figure(figsize=(7, 6))
plt.pcolormesh(sand)
plt.colorbar()
plt.show
# now let's play
```

```
""" #only adds sand if whole board is stable
for i in range(1000):
    sand[49,49]+=1
    sand_1 = play_sand(sand)
    if (sand_1==sand).all():
        sand[49,49]+=1
    else:
        sand = sand_1
    """
# adds every second

for i in range(5000):
    sand[49,49]+=1
    sand_1 = play_sand(sand)
    if (sand_1==sand).all():
        sand[49,49]+=1
    else:
        sand = sand_1
plt.figure(figsize=(7, 6))
plt.pcolormesh(sand)
plt.colorbar()
plt.show
```

Hormiga de langton

```
GRID_SIZE = (160, 120)
GRID_SQUARE_SIZE = (4, 4)
ant_image_filename = "blue-dot.png"
ITERATIONS = 10

import pygame
from pygame.locals import *

class AntGrid(object):

    def __init__(self, width, height):

        self.width = width
        self.height = height
```

```
        self.clear()

    def clear(self):

        self.rows = []
        for col_no in xrange(self.height):
            new_row = []
            self.rows.append(new_row)
            for row_no in xrange(self.width):
                new_row.append(False)

    def swap(self, x, y):
        self.rows[y][x] = not self.rows[y][x]

    def get(self, x, y):
        return self.rows[y][x]

    def render(self, surface, colors, square_size):

        w, h = square_size
        surface.fill(colors[0])

        for y, row in enumerate(self.rows):
            rect_y = y * h
            for x, state in enumerate(row):
                if state:
                    surface.fill(colors[1], (x * w, rect_y, w, h))


class Ant(object):

    directions = ( (0,-1), (+1,0), (0,+1), (-1,0) )

    def __init__(self, grid, x, y, image, direction=1):

        self.grid = grid
        self.x = x
        self.y = y
```

```
        self.image = image
        self.direction = direction


    def move(self):

        self.grid.swap(self.x, self.y)

        self.x = ( self.x + Ant.directions[self.direction][0] ) % self.grid.width
        self.y = ( self.y + Ant.directions[self.direction][1] ) % self.grid.height

        if self.grid.get(self.x, self.y):
            self.direction = (self.direction-1) % 4
        else:
            self.direction = (self.direction+1) % 4


    def render(self, surface, grid_size):

        grid_w, grid_h = grid_size
        ant_w, ant_h = self.image.get_size()
        render_x = self.x * grid_w - ant_w / 2
        render_y = self.y * grid_h - ant_h / 2
        surface.blit(self.image, (render_x, render_y))


def run():

    pygame.init()
    pygame.display.set_caption('hormigas langton')
    w = GRID_SIZE[0] * GRID_SQUARE_SIZE[0]
    h = GRID_SIZE[1] * GRID_SQUARE_SIZE[1]
    screen = pygame.display.set_mode((w, h), 0, 32)

    ant_image = pygame.image.load(ant_image_filename).convert_alpha()

```

```
    default_font = pygame.font.get_default_font()
    font = pygame.font.SysFont(default_font, 10)

    ants = []
    grid = AntGrid(*GRID_SIZE)
    running = False

    total_iterations = 0

    while True:

        for event in pygame.event.get():

            if event.type == QUIT:
                return

            if event.type == MOUSEBUTTONDOWN:

                x, y = event.pos
                x /= GRID_SQUARE_SIZE[0]
                y /= GRID_SQUARE_SIZE[1]

                ant = Ant(grid, int(x), int(y), ant_image)
                ants.append(ant)


            if event.type == KEYDOWN:

                if event.key == K_SPACE:
                    running = not running

                if event.key == K_c:
                    grid.clear()
                    total_iterations = 0
                    del ants[:]


        grid.render(screen, ((0, 0, 0), (255, 255, 255)), GRID_SQUARE_SIZE)
```

```

        if running:
            for iteration_no in xrange(ITERATIONS):
                for ant in ants:
                    ant.move()
            total_iterations += ITERATIONS

        txt = "%i"%total_iterations
        txt_surface = font.render(" %i "%total_iterations, True, (120,
    120,120))
        screen.blit(txt_surface, (0, 0))

        for ant in ants:
            ant.render(screen, GRID_SQUARE_SIZE)


        pygame.display.flip()
        pygame.time.delay(10)


if __name__ == "__main__":
    run()
```

Appendix B

Tema 2: Códigos

B.1. Grafos completos

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 18 19:16:07 2018

@author: booort
"""
#import libraries
import networkx as nx
import matplotlib.pyplot as plt

#Function that show and save the first n completed graphs
def complete_graph_generator(n):
    #for-loop for printing and save all graphs generated
    for i in range(1,n+1):
        G=nx.complete_graph(i)
        nx.draw_networkx(G)
        plt.savefig('grafo_completo_grado_%i.png' %(i))
        plt.show()
#I've used networkx library, de code that generate the graph is:
    """
```

```
    def complete_graph(n,create_using=None):
    ###
    Return the complete graph K_n with n nodes.

    Node labels are the integers 0 to n-1.
    ###
    G=empty_graph(n,create_using)
    G.name="complete_graph( %d)" %(n)
    if n>1:
        if G.is_directed():
            edges=itertools.permutations(range(n),2)
        else:
            edges=itertools.combinations(range(n),2)
        G.add_edges_from(edges)
    return G
    """
#finally we call the function
complete_graph_generator (8)
```

Appendix C

Tema 3: Códigos

C.1. Hodgkin–Huxley modelo

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Hodgkin-Huxley Model

@author: booort
"""

import scipy as sp
import pylab as plt
from scipy.integrate import odeint

# Constantes
C_m = 1.0
g_l = 0.3
g_K = 36.0
g_Na = 120.0
V_l = -54.402
V_K = -77.0
V_Na = 50.0

```

```

def alpha_m(V):
    return 0.1*(V+40.0)/(1.0 - sp.exp(-0.1*(V+40.0)))


def alpha_n(V):
    return 0.01*(V+55.0)/(1.0 - sp.exp(-0.1*(V+55.0)))


def beta_h(V):
    return 1.0/(1.0 + sp.exp(-0.1*(V+35.0)))


def alpha_h(V):
    return 0.07*sp.exp(-0.05*(V+65.0))


def beta_m(V):
    return 4.0*sp.exp(-0.0556*(V+65.0))


def beta_n(V):
    return 0.125*sp.exp(-0.0125*(V+65))


#Definicion de F

def I_Na(V, m, h):
    return g_Na*m**3*h*(V-V_Na)


def I_K(V, n):
    return g_K*n**4*(V-V_K)


def I_L(V):
    return g_l*(V-V_l)

```

```
#  suma de la corrientes
#  externas y sinapticas entrando en la celula, cada una de ellas por unidad de
#  area de la membrana celular
def I_inj(t):
    return 10*(t>20) - 10*(t>40) + 5*(t>65) - 5*(t>85) + 2.5*(t>105) - 2.5*(t>125) + 2.2*(t>140) - 2.2*(t>160)


#Tiempo donde vamos a integrar
t = sp.arange(0.0, 200.0, 0.1)

# Integracion numerica
def dALLdt(X, t):
    V, m, h, n = X

    dVdt = (I_inj(t) - I_Na(V, m, h) - I_K(V, n) - I_L(V)) / C_m
    dmdt = alpha_m(V)*(1.0-m) - beta_m(V)*m
    dhdt = alpha_h(V)*(1.0-h) - beta_h(V)*h
    dndt = alpha_n(V)*(1.0-n) - beta_n(V)*n
    return dVdt, dmdt, dhdt, dndt

X = odeint(dALLdt, [-65, 0.05, 0.6, 0.32], t)

#  Rescatamos el valor que nos interesa teniendo en cuenta el output de odeint

V = X[:,0]

plt.figure()
plt.plot(t, V, label=r'V (mV)')
plt.plot(t, I_inj(t),'r', label=r'I_iny ($\mu A/cm^2$)')
plt.xlabel('Time(msec)')
plt.legend(loc='upper right' )

plt.show()

plt.plot(V, (alpha_m(V)/(alpha_m(V)+beta_m(V))), label=r'V (mV)')
```

```
plt.show()
```

C.2. FitzHugh–Nagumo modelo

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue May 29 10:25:30 2018

@author: booort
"""
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import scipy as sp


def F_N_Model(X, t, phi, alpha, betta, I_iny):
    V, U = X

    dVdt = V-V**3/3-U+I_iny
    dUdt = phi*(V+alpha-betta*U)
    return dVdt, dUdt

def V_nullcline(V, I_iny):
    return V - 1./3*V**3 + I_iny

def U_nullcline(U,alpha, betta):
    return -alpha + betta*U

#Cte values from wikipedia
alpha=0.7
betta=0.8
phi=0.08
I_iny=[0.2,0.4,0.8,2.0]

for i in range(0,4):

```

```
    t = np.arange(0, 100, 0.1)
    X_0=[-0.5,-0.5]
    sol = odeint(F_N_Model, X_0, t,args=(phi,alpha,betta,I_iny[i]))

    # calculo y resolucion de las nulclinas
    x = sp.arange(-3, 3, 0.1)
    y = sp.arange(-4, 4, 0.1)
    v_nul= V_nullcline(x,I_iny[i])
    u_nul = U_nullcline(y,alpha,betta)


    #plot de las nulclinas
    fig, ax = plt.subplots(figsize=(6,9))
    ax.set_title("inyecte I value = %i" %I_iny[i])
    plt.subplot(211)
    plt.rc('grid', linestyle=":", color='black')
    plt.plot(x,v_nul,'grey',linestyle="--",linewidth=2,label='nulclina V')
    plt.plot(u_nul,y,'grey',linestyle=":",linewidth=2,label='nulclina U')
    plt.ylabel('u')
    plt.xlabel('v')
    #plot de la solucion
    plt.plot(sol[:, 0],sol[:, 1],  'b', label='trayectoria')
    plt.legend(loc='best')

    plt.grid()
    ax.set_title("inyecte I value = %i" %I_iny[i])
    plt.subplot(212)
    plt.plot(t,sol[:, 0],  'b', label='V')
    plt.xlabel('t')
    plt.legend(loc='best')
    ax.set_title("inyecte I value = %i" %I_iny[i])
    plt.show()
```

C.3. Tsodyks-Markram modelo

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Tue May 29 12:10:16 2018

@author: booort
"""

import scipy as sp
import pylab as plt
from scipy.integrate import odeint
import numpy as np
#constantes del articulo original de tsodysk
tau_rec = 800
tau_in = 3
tau_m =40
R_in = 500
A_SE = 150
t = np.arange(0, 100, 0.1)

def delta_(t):
    return 10*(t>9) - 10*(t>10) + 10*(t>12) - 10*(t>13) + 10*(t>16) - 10*(t
    >17) + 10*(t>20) - 10*(t>21) + 10*(t>24) - 10*(t>25)+ 10*(t>59) - 10*(t
    >60)+ 10*(t>69) - 10*(t>70)+ 10*(t>79) - 10*(t>80)+ 10*(t>89) - 10*(t
    >90)+ 10*(t>99) - 10*(t>100)


def T_M_Model(X, t, tau_in, tau_m, R_in, A_SE):
    x,y,z,V = X

    dxdt = z/tau_rec-V*x*delta_(t)
    dydt = -y/tau_in-V*x*delta_(t)
    dzdt = y/tau_in-z/tau_rec
    dVdt = 1/tau_m*(-V+R_in*A_SE*y)
    return dxdt,dydt,dzdt,dVdt

X_0=[0.02,0.02,0.02,0]
sol = odeint(T_M_Model, X_0, t, args=(tau_in, tau_m, R_in, A_SE))
V=sol[:,3]
x=sol[:,0]
y=sol[:,1]
```

```
z=sol[:,2]
plt.plot(t,V)
plt.plot(t,delta_(t))
plt.show()
```

Appendix D

Tema 4: Códigos

D.1. Voter model con Barabasi-Albert

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  6 12:22:19 2018

@author: booort
"""

import networkx as nx
import random as rn
import matplotlib.pyplot as plt

#number of steps
run_time=100000

#variables for de B-A graph creation, and graoh creation
nodes=1000
edges=2
population=nx.barabasi_albert_graph(nodes,edges)

#We set all nodes with 'r' variable and print a summary for our graph
```

```
nx.set_node_attributes(population,'r','partido')
print(nx.info(population))

#visual representaiton
nx.draw(population)
plt.show()
#We randomize the partido variable between nodes
changed_nodes=[]
not_changed=[]
for i in range(len(population.nodes())):
    aux=rn.randrange(0,2,1)
    if aux==1:
        population.node[i]['partido']='b'
        changed_nodes.append(i)
    else:
        not_changed.append(i)

#visual representation after the randomize
   # we get the position of the nodes
pos=nx.spring_layout(population)
nx.draw_networkx_nodes(population,pos,
                       nodelist=changed_nodes,
                       node_color='b',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_nodes(population,pos,
                       nodelist=not_changed,
                       node_color='r',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_edges(population,pos,width=1.0,alpha=0.5)
plt.show()
print('Inicialmente tenemos {} azules y {} rojos'.format(len(changed_nodes),len(not_changed)))

#main part: we execute voter model 'run_time' times
vector_proporcion=[]
for i in range(run_time):
```

```
    node_influencer=rn.randrange(0, nodes-1,1)
    node_receiver=list(population.neighbors(node_influencer))[rn.randrange
    (0, len(list(population.neighbors(node_influencer))),1)]
    if population.node[node_influencer]['partido']!=population.node[
    node_receiver]['partido']:
        if population.node[node_influencer]['partido']=='r':
            population.node[node_receiver]['partido']='r'
        else:
            population.node[node_receiver]['partido']='b'
    result_changed=[]
    result_not_changed=[]
    for j in range(len(population.nodes())):
        if population.node[j]['partido']=='b':
            result_changed.append(j)
        else:
            result_not_changed.append(j)
    vector_proporcion.append(len(result_changed)/len(result_not_changed))



#visual representation of the resfult

pos=nx.spring_layout(population)
nx.draw_networkx_nodes(population,pos,
                       nodelist=result_changed,
                       node_color='b',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_nodes(population,pos,
                       nodelist=result_not_changed,
                       node_color='r',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_edges(population,pos,width=1.0,alpha=0.5)
plt.show()


fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
ax.plot(vector_proporcion)
ax.set_xscale('log')
ax.set_yscale('log')
plt.show()
```

D.2. Voter model con Small-World

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  6 14:32:23 2018

@author: booort
"""

import networkx as nx
import random as rn
import matplotlib.pyplot as plt

#number of steps
run_time=100000

#variables for de B-A graph creation, and graoh creation
nodes=100
edges=3
probability=1
population=nx.watts_strogatz_graph(nodes,edges,probability)

#We set all nodes with 'r' variable and print a summary for our graph
nx.set_node_attributes(population,'r','partido')
print(nx.info(population))

#visual representaiton
nx.draw(population)
plt.show()

#We randomize the partido variable between nodes
```

```
changed_nodes=[]
not_changed=[]
for i in range(len(population.nodes())):
    aux=rn.randrange(0,2,1)
    if aux==1:
        population.node[i]['partido']='b'
        changed_nodes.append(i)
    else:
        not_changed.append(i)

#visual representation after the randomize
   # we get the position of the nodes
pos=nx.spring_layout(population)
nx.draw_networkx_nodes(population,pos,
                       nodelist=changed_nodes,
                       node_color='b',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_nodes(population,pos,
                       nodelist=not_changed,
                       node_color='r',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_edges(population,pos,width=1.0,alpha=0.5)
plt.show()
print('Inicialmente tenemos {} azules y {} rojos'.format(len(changed_nodes),len(not_changed)))

#main part: we execute voter model 'run_time' times
vector_proporcion=[]
for i in range(run_time):
    node_influencer=rn.randrange(0, nodes-1,1)
    node_receiver=list(population.neighbors(node_influencer))[rn.randrange(0, len(list(population.neighbors(node_influencer))),1)]
    if population.node[node_influencer]['partido']!=population.node[node_receiver]['partido']:
        if population.node[node_influencer]['partido']=='r':
            population.node[node_receiver]['partido']='r'
```

```
        else:
            population.node[node_receiver]['partido']='b'
    result_changed=[]
    result_not_changed=[]
    for j in range(len(population.nodes())):
        if population.node[j]['partido']=='b':
            result_changed.append(j)
        else:
            result_not_changed.append(j)
    vector_proporcion.append(len(result_changed)/len(result_not_changed))


#visual representation of the resfult

pos=nx.spring_layout(population)
nx.draw_networkx_nodes(population,pos,
                       nodelist=result_changed,
                       node_color='b',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_nodes(population,pos,
                       nodelist=result_not_changed,
                       node_color='r',
                       node_size=100,
                   alpha=0.8)
nx.draw_networkx_edges(population,pos,width=1.0,alpha=0.5)
plt.show()


fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(vector_proporcion)
ax.set_xscale('log')
ax.set_yscale('log')
plt.show()
```