Estructura celular y patrones en biología: un enfoque de biología de sistemas

Anexos

A. Notación

La notación a utilizar durante el tratamiento teórico de las ecuaciones de regulación, y el significado de cada parámetro se presenta en la siguiente tabla.

Símbolo	Significado
∇^2	Operador 'Laplaciano'.
C_i	Concentración de la proteína i en la célula observada.
D_i	Coeficiente de difusión de la proteína i.
μ_i	Coeficiente de degradación de la proteína i
H_i^j	Coeficiente multiplicativo ('amplitud') de la función de Hill que representa la influencia de j sobre i.
K_i^j	Valor umbral de C_j en la función de Hill que representa la influencia de j sobre i.
n_i^j	Exponente de Hill de la función de Hill que representa la influencia de j sobre i.
$h_i^j = \frac{(C_j)^{n_i^j}}{(K_i^j)^{n_i^j}+(C_j)^{n_i^j}}$	Función de Hill (normalizada) de activación ($j \to i$) que representa la influencia de j sobre i.
$\bar{h}_i^j = 1 - h_i^j$	Función de Hill (normalizada) de represión ($j \dashv i$) que representa la influencia de j sobre i.
i'	Concentración de i en el instante $t - \delta t$. Lo usamos para denotar las interacciones con retardo.
$\hat{i}$	Promedio de las concentraciones de i en las células vecinas. Lo usamos para denotar las interacciones a través de ligandos.
$\tilde{X}$	Parámetro adimensional con la función de X.
A_i	Concentración inicial de i para las células consideradas en la parte anterior al MF.
P_i	Concentración inicial de i para las células consideradas en la parte posterior al MF.

Tabla 4: Descripción y significado de los distintos símbolos a utilizar en nuestras ecuaciones de dinámica.

B. Adimensionalización

Tenemos 7 grados de libertad en la totalidad del sistema. De la arbitrariedad en las unidades de C_i tenemos 6, y el séptimo lo aporta la escala de tiempo. Las dimensiones espaciales no aportan ningún grado de libertad por que se escala con el tamaño de las células.

La primera adimensionalización la realizamos tomando una nueva variable temporal $\tau = \mu_1$, de forma que nos eliminamos el parámetro de degradación de Hh:

$$\frac{\partial C_i}{\partial \tau} = \frac{D_i}{\mu_1}\nabla^2 C_i + \sum_j^A \frac{H_i^j}{\mu_1} h_i^j + \sum_j^I \frac{H_i^j}{\mu_1}\bar{h}_i^j - \frac{\mu_i}{\mu_1} C_i \tag{B.1}$$

A parte de esto, también aproximamos $\mu_i = \mu_1 \; \forall i \Rightarrow \tilde{\mu}_i = 1 \forall i$. De esta forma tendremos:

$$\frac{\partial C_i}{\partial \tau} = \frac{D_i}{\mu_1}\nabla^2 C_i + \sum_j^A \frac{H_i^j}{\mu_1} h_i^j + \sum_j^I \frac{H_i^j}{\mu_1}\bar{h}_i^j - C_i \tag{B.2}$$

Ahora, para eliminar los grados de libertad de las concentraciones, lo que nos hemos propuesto es que el umbral (K_i^j) de la función de Hill con $H_i^j \neq 0$ y menor j pase a ser $\tilde{K}_i^j = 1$. Si definimos el K_i^j previo como K_i^{ref}, con ref dependiente de i como ya hemos dicho, tenemos, sustituyendo en las funciones de Hill $\tilde{C}_i = C_i / K_i^{ref}$:

$$K_i^{ref}\frac{\partial \tilde{C}_i}{\partial \tau} = \frac{D_i}{\mu_1}\left(K_i^{ref}\right)^2\nabla^2 \tilde{C}_i + \sum_j^A \frac{H_i^j}{\mu_1} h_i^j + \sum_j^I \frac{H_i^j}{\mu_1}\bar{h}_i^j - K_i^{ref}\tilde{C}_i \tag{B.3}$$

donde las funciones de Hill expresadas en términos de K_i^{ref} son:

$$h_i^j = \frac{\left(\tilde{C}_i\right)^n}{\left(K_i^j/K_i^{ref}\right)^n + \left(\tilde{C}_i\right)^n} \qquad \bar{h}_i^j = \frac{\left(K_i^j/K_i^{ref}\right)^n}{\left(K_i^j/K_i^{ref}\right)^n + \left(\tilde{C}_i\right)^n} \tag{B.4}$$

Si dividimos ahora todo entre K_i^{ref} llegamos a la expresión final adimensionalizada:

$$\frac{\partial \tilde{C}_i}{\partial \tau} = \tilde{D}_i\nabla^2 \tilde{C}_i + \sum_j^A \tilde{H}_i^j \frac{\left(\tilde{C}_i\right)^n}{\left(\tilde{K}_i^j\right)^n + \left(\tilde{C}_i\right)^n} + \sum_j^I \tilde{H}_i^j \frac{\left(\tilde{K}_i^j\right)^n}{\left(\tilde{K}_i^j\right)^n + \left(\tilde{C}_i\right)^n} - \tilde{C}_i \tag{B.5}$$

Las relaciones entre las variables originales y las adimensionalizadas son:

- $\tau = \mu_1 t$
- $\tilde{C}_i = \frac{1}{K_i^{ref}} C_i$
- $\tilde{D}_i = \frac{K_i^{ref}}{\mu_1} D_i$
- $\tilde{H}_i^j = \frac{1}{\mu_1 K_i^{ref}} H_i^j$
- $\tilde{K}_i^j = \frac{1}{K_i^{ref}} K_i^j$

C. Parámetros utilizados

En esta sección vamos a presentar en formato de tabla todos los parámetros utilizados en las simulaciones cuyos resultados se han presentado. En cada una de ellas mostraremos únicamente los que sean relevantes para cada caso. En todas ellas se han impuesto las mismas condiciones de contorno. Estas consisten en asignar a un cuarto de las células del tejido (parte izquierda) las concentraciones correspondientes a la parte posterior al MF. Al resto se les asignan las correspondientes a la parte anterior.

Todos los parámetros expuestos son adimensionales, sin embargo, para no sobrecargar la notación, los denotamos por sus símbolos 'originales'.

C.1. Modelo simple

Parámetro	D_1	H_1^1	H_3^1	A_1	A_3	P_1	P_3
Valor	1	6	6	0	6	6	0

Tabla 5: Parámetros de la simulación que involucra a Hh y *Hairy*. Ésta es la presentada en la Figura 17.

Parámetro	D_1	D_2	H_1^1	H_2^1	H_3^1	H_3^2	A_1	A_2	A_3	P_1	P_2	P_3
Valor	1	30	6	6	6	0,5	0	0	6	6	6	0,5

Tabla 6: Parámetros de la simulación que involucra a Hh, Dpp y *Hairy*. Ésta es la presentada en la Figura 18.

Parámetro	D_1	H_3^1	H_4^1	H_4^3	H_1^4	A_1	A_3	A_4	P_1	P_3	P_4
Valor	1	6	4	2	6	0	6	0	6	0	6

Tabla 7: Parámetros de la simulación que involucra a Hh, *Hairy* y Ato. Ésta es la presentada en la Figura 20.

Parámetro	D_1	D_2	H_2^1	H_3^1	H_4^1	H_3^2	H_4^2	H_4^3	H_1^4
Valor	1	30	6	6	3,5	1	1,5	2	6

Tabla 8: Parámetros dinámicos de la simulación que involucra a Hh, Dpp, *Hairy* y Ato. Ésta es la presentada en la Figura 21.

Parámetro	A_1	A_2	A_3	A_4	P_1	P_2	P_3	P_4
Valor	0	0	6	0	6	6	1	6

Tabla 9: Condiciones iniciales de la simulación que involucra a Hh, Dpp, *Hairy* y Ato. Ésta es la presentada en la Figura 21.

C.2. Modelo *Delta-Notch*

Presentamos aquí los parámetros que utilizados en la simulación cuyos resultados se presentan en la Figura 22. Dado el gran número de estos parámetros, hemos optado por indicar en tablas separadas los distintos parámetros.

Parámetro	H_2^1	H_3^1	H_4^1	H_5^1	H_3^2	H_4^2	H_5^2	H_4^3	H_1^4	H_3^5	H_3^6	H_4^6	H_5^6	$H_6^{\hat{5}}$
Valor	2,5	2,5	2,5	0,5	0,5	2	0,6	1	2,5	1	1	1	2,9	100

Tabla 10: Parámetros H_i^j de la simulación del sistema básico junto con el Dl-N. Recordamos que $\hat{i}$ representa el promedio de la concentración de i en las células vecinas.

Parámetro	D_1	D_2	K_3^2	K_4^2	K_5^2	K_4^3	K_3^5	K_3^6	K_4^6	K_5^6
Valor	0,8	1,5	0,2	1	2	1	1	1	1	20

Tabla 11: Parámetros K_i^j de la simulación del sistema básico junto con el Dl-N.

Parámetro	A_1	A_2	A_3	A_4	A_5	A_6
Valor	0	0	2,5	0	0	0

Tabla 12: Condiciones iniciales en la parte anterior (A_i^j) al MF de la simulación del sistema básico junto con el Dl-N.

Parámetro	P_1	P_2	P_3	P_4	$P_5\|_{Dl}$	$P_5\|_N$	$P_6\|_{Dl}$	$P_6\|_N$
Valor	2,31	2,31	0,5	2,8	3,5	0,8	0	50

Tabla 13: Condiciones iniciales en la parte posterior (P_i^j) al MF de la simulación del sistema básico junto con el Dl-N. Notar que dado que comenzamos con el patrón ya formado debemos diferenciar las concentraciones de Dl y N en según el tipo de célula. Aquí $P_5|_N$ representa la concentración inicial de '5' (Dl) en una célula 'tipo' *Notch* de la parte posterior al MF. El resto deben interpretarse de la misma forma.

C.3. Modelo de inhibición con retardo

En este apartado adjuntamos el valor de los parámetros utilizados en los diagramas de fase de la Figura 23, y los de la simulación de la Figura 26. En este último caso habrá un nuevo parámetro que considerar, el retardo en la inhibición (Hh⊣Ato), que denotaremos $\delta\tau$.

Parámetro	H_4^1	H_1^4	$H_4^{1'}$	$K_4^{1'}$	n
Valor	1,4	1,4	1,1	0,5	∞

Tabla 14: Parámetros característicos de la Figura 23 (A). Recordamos que los parámetros primados hacen referencia a la interacción retardada.

Parámetro	H_4^1	H_1^4	$H_4^{1'}$	$K_4^{1'}$	n
Valor	4	1,27	1,1	0,1	3

Tabla 15: Parámetros característicos de la Figura 23 (B). Recordamos que los parámetros primados hacen referencia a la interacción retardada.

Presentamos por último los parámetros correspondientes a la simulación cuyos resultados son presentados en la Figura 26.

Parámetro	$\delta\tau$	D_1	D_2	H_2^1	H_4^1	H_4^2	H_1^4	$H_4^{1'}$	K_4^2	$K_4^{1'}$
Valor	1	0,8	1,5	2	4	0,5	1,27	1,1	0,2	0,1

Tabla 16: Parámetros característicos de la dinámica de la simulación cuyos resultados se exponen en la Figura 26. Recordamos que los parámetros primados hacen referencia a la interacción retardada.

Parámetro	A_1	A_2	A_4	P_1	P_2	P_4
Valor	0,1	0	0,5	0,7	0	1

Tabla 17: Condiciones iniciales de la simulación que involucra a Hh, Dpp y Ato considerando la inhibición retardada. Éstas son las usadas para la simulación de la Figura 26

D. Programas

En este anexo vamos a exponer los programas que principalmente utilizamos durante el trabajo. Los que recogen los modelos ya presentados son los 'Simuladores' en 1D (Sección D.1) y 2D (Sección D.2), que bajo los parámetros escogidos recogen los resultados de la simulación (tiempo, número de célula, concentraciones...) en documentos de texto. Posteriormente estos documentos de texto se leen por los programas 'Representación' (Secciones D.4 y D.5, que 'llaman' a *Gnuplot* para representar los datos. La representación (para poder extraer fácilmente comportamientos del sistema) se realiza como una sucesión de imágenes que evolucionan conforme el programa las lee y dibuja (o más lento si se desea).

Las notaciones en los distintos códigos y su funcionamiento varían levemente. En el *Simulador 1D* (Sección D.1), al estar involucradas menos proteínas nos permitimos una notación más laxa, llamando a las distintas variables en función de sus relaciones con las proteínas (Hh, Dpp...) y no sus índices (1, 2...). Tampoco denotamos las condiciones iniciales en los programas de la misma manera en que lo hacemos en el trabajo. Sin embargo, en cada uno de los códigos se explican con anotaciones los significados de las distintas variables y funciones.

El funcionamiento de los programas de simulación consiste en integrar numéricamente las ecuaciones de la dinámica mediante el algoritmo de *Euler*. Este algoritmo nos permite observar la evolución del sistema en el tiempo. Los pasos de tiempo utilizados para las simulaciones cubren un rango aproximado $d\tau \in (0{,}01 - 0{,}0001)$. En algunos casos hemos introducido fluctuaciones en las concentraciones, para ello hemos utilizado el generador de números aleatorios '*Parisi-Rapuano*' en conjunto con el algoritmo de '*Box Muller*' para obtener una distribución gaussiana.

En algunos programas puede observarse implementada alguna de las proteínas que en el trabajo habíamos comentado que eran redundantes (Emc en el Anexo D.1). Concluimos que ésta era redundante con *Hairy* y la descartamos para posteriores simulaciones. Por otro lado, en el Anexo D.2 también se contempla una inhibición retardada de Ato debido a *Notch* (que también se propone como mediadora en la activación de Sca, el inhibidor de Ato, por Hh) [16, 18]. Dado que ambas modificaciones no alteraban de forma decisiva el comportamiento del sistema, se descartaron rápidamente. En el Anexo D.2 usamos el mismo formato de numeración de células (presentado en la Figura 27) que P.J. Blasco Hernández en [1].

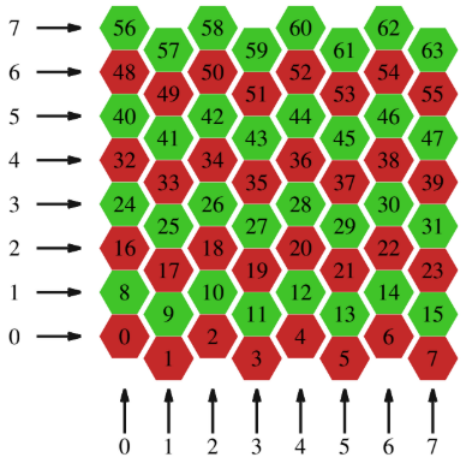

Figura 27: El diagrama indica la numeración de cada una de las células. Imagen tomada de [1]

D.1. Simulador 1D

```
//LIBRERIAS
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//---PARÁMETROS DEL PROGRAMA---

//Herramientas
#define PROGRESO //Marca como transcurre la simulación.
#define res 1 //Resolución en porcentaje

//Parámetros referentes al sistema y simulación
#define N 400 //Dimensión del sistema
#define dt 0.001 //Paso temporal
#define ts 2 //Tiempo de simulación
#define rdi 1000 /*Relación de datos calculados frente a escritos
(se escribe 1 por cada 'rdi' calculados)*/

//Parámetros referentes a las propiedades físicas
/*Notación
    DX ---> coef. de difusión de X
    CXY --> coef. de dependencia para Y(X)
    uX ---> coef. de degradación de X
*/
#define DHh 1 //Hedgehog
#define DDpp 30 //Decapentaplegic

#define uHh 1
#define uDpp 1
#define uH 1 //Hairy
#define uEmc 1 //Extramacrochaete
#define uAto 1 //Atonal

#define CHhHh 6
#define CAtoHh 0
#define CHhDpp 6
#define CDppH 0.5
#define CHhH 6
#define CDppEmc 0
#define CHhEmc 0
#define CHhAto 0
#define CDppAto 0
#define CHAto 0
#define CEmcAto 0

#define n 3 //Exponente de las funciones de Hill

/*Condiciones iniciales
```

```
    CI --> Condición
    0 ---> Perturbación sobre primera célula
    1 ---> Perturbación sobre célula más central
    2 ---> Forma exponencial desde la célula central
    3 ---> Perturba el primer 1/4 de la posición
*/
#define ci 3

#define Hh0 0
#define Dpp0 0
#define H0 6
#define Emc0 0
#define Ato0 0

#define dHh0 6
#define dDpp0 6
#define dH0 -5.5
#define dEmc0 0
#define dAto0 6

//SUBALGORITMOS
void CI(double *Hh, double *Dpp, double *H, double *Emc, double *Ato);
//Genera el estado inicial del sistema
void Registra(double t, double *Hh, double *Dpp, double *H, double *Emc,
double *Ato);//Guarda la configuración actual
void Evolucion(double *Hh, double *Dpp, double *H, double *Emc,
double *Ato);//Avanza un paso temporal
void Laplaciano(double *f, double *Lf);//Calcula el Laplaciano de 'f'
void Velocidades(double *Hh, double *LHh, double *Dpp, double *LDpp, double *H,
double *Emc, double *Ato, double *vHh, double *vDpp, double *vH, double *vEmc,
double *vAto);//Calcula el cambio de concentración con el tiempo (de todas)
double hill(double a);//Función de Hill de activación
double nhill(double a);//Función de Hill de inhibición

//Variables de uso global
FILE *DATA;

//CÓDIGO PRINCIPAL
int main()
{
    FILE *info;//guardamos los datos utilizados para variables generales
    info=fopen("INFO.txt","w");
    fprintf(info,"%d %d %d",int(ts/dt/rdi),int(N),int(ci));
    fclose(info);
    //Definimos nuestras variables
    double Hh[N],Dpp[N],H[N],Emc[N],Ato[N],t;
    int i,Nt;
    /*---DATOS---

        X[i] --> Concentración de 'X' en la posición 'i'
        t -----> Tiempo transcurrido en la simulación
```

```
        Nt ----> Nº de pasos temporales a simular
    */
    Nt=int(double(ts)/dt);
    DATA=fopen("Datos[t,x,Hh,Dpp,Hairy,Emc,Ato].txt","w");/*Nombramos el
    fichero en el que vamos a escribir los datos*/
    CI(Hh,Dpp,H,Emc,Ato);//Establecemos las condiciones iniciales
    t=0;//inicializamos el tiempo
    for(i=0;i<Nt+1;i++)
    {
        if(i%int(rdi)==0)
            Registra(t,Hh,Dpp,H,Emc,Ato);
        t+=dt;
        Evolucion(Hh,Dpp,H,Emc,Ato);
        #ifdef PROGRESO
            if(100*int(res)*i%Nt==0)
                printf("%d%c\n",100*int(res)*i/Nt,'%');
        #endif // PROGRESO
    }
    fclose(DATA);
    DATA=fopen("Datos(fin)[t,x,Hh,Dpp,Hairy,Emc,Ato].txt","w");
    Registra(t,Hh,Dpp,H,Emc,Ato);
}

//SUBALGORITMOS
void CI(double *Hh, double *Dpp, double *H, double *Emc, double *Ato)
{
    int i;
    for(i=0;i<N;i++)
    {
        Hh[i]=Hh0;
        Dpp[i]=Dpp0;
        H[i]=H0;
        Emc[i]=Emc0;
        Ato[i]=Ato0;
    }
    switch(ci)
    {
    case 0:
        Hh[0]+=dHh0;
        Dpp[0]+=dDpp0;
        H[0]+=dH0;
        Emc[0]+=dEmc0;
        Ato[0]+=dAto0;
    break;
    case 1:
        Hh[int(N)/2]+=dHh0;
        Dpp[int(N)/2]+=dDpp0;
        H[int(N)/2]+=dH0;
        Emc[int(N)/2]+=dEmc0;
        Ato[int(N)/2]+=dAto0;
    break;
```

```
    case 2:
        for(i=0;i<N;i++)
        {
            Hh[i]+=dHh0*exp(-(i-int(N/2))*(i-int(N/2))/DHh);
            Dpp[i]+=dDpp0*exp(-(i-int(N/2))*(i-int(N/2))/DDpp);
            Ato[i]+=dAto0*exp(-(i-int(N/2))*(i-int(N/2))/DHh);
        }
    break;
    case 3:
        for(i=0;i<N/4;i++)
        {
            Hh[i]+=dHh0;
            Dpp[i]+=dDpp0;
            H[i]+=dH0;
            Emc[i]+=dEmc0;
            Ato[i]+=dAto0;
        }
    break;
    default:
        printf("No se ha seleccionado correctamente la condición inicial,"
        "se toma el caso '1' por defecto\n\n");
        Hh[int(N)/2]+=dHh0;
        Dpp[int(N)/2]+=dDpp0;
        H[int(N)/2]+=dH0;
        Emc[int(N)/2]+=dEmc0;
        Ato[int(N)/2]+=dAto0;
    }
    Registra(0,Hh,Dpp,H,Emc,Ato);
}

void Registra(double t, double *Hh, double *Dpp, double *H, double *Emc,
double *Ato)//Guarda la configuración actual
{
    int i;
    for(i=0;i<N;i++)
    {
        fprintf(DATA,"%lf   %d  %lf %lf %lf %lf %lf\n",t,i,Hh[i],Dpp[i],
        H[i],Emc[i],Ato[i]);
    }
}

void Evolucion(double *Hh, double *Dpp, double *H, double *Emc,
double *Ato)
{
    double LHh[N],LDpp[N],vHh[N],vDpp[N],vH[N],vEmc[N],vAto[N];/*Formamos
    el Laplaciano de f y la primera derivada temporal de ambas variables.*/
    int i;
    Laplaciano(Hh,LHh);//Calculamos el Laplaciano de Hh
    Laplaciano(Dpp,LDpp);//Calculamos el Laplaciano de Dpp
    Velocidades(Hh,LHh,Dpp,LDpp,H,Emc,Ato,vHh,vDpp,vH,vEmc,vAto);
    for(i=1;i<N-1;i++)
```

```
    {
        Hh[i]+=vHh[i]*dt;
        Dpp[i]+=vDpp[i]*dt;
        H[i]+=vH[i]*dt;
        Emc[i]+=vEmc[i]*dt;
        Ato[i]+=vAto[i]*dt;
    }
}

void Laplaciano(double *f, double *Lf)
{
    int i;
    Lf[0]=f[1]-2*f[0];
    Lf[N]=f[N-1]-2*f[N];
    for(i=1;i<N-1;i++)
        Lf[i]=f[i-1]+f[i+1]-2*f[i];
}

void Velocidades(double *Hh, double *LHh, double *Dpp, double *LDpp,
double *H, double *Emc, double *Ato, double *vHh, double *vDpp,
double *vH, double *vEmc, double *vAto)
{
    int i;
    for(i=0;i<N;i++)
    {
        vHh[i]=DHh*LHh[i]+CHhHh*hill(Hh[i])+CAtoHh*hill(Ato[i])-uHh*Hh[i];
        vDpp[i]=DDpp*LDpp[i]+CHhDpp*hill(Hh[i])-uDpp*Dpp[i];
        vH[i]=CDppH*hill(Dpp[i])+CHhH*nhill(Hh[i])-uH*H[i];
        vEmc[i]=CDppEmc*hill(Dpp[i])+CHhEmc*nhill(Hh[i])-uEmc*Emc[i];
        /*vAto[i]=(CHhAto*hill(Hh[i])+CDppAto*hill(Dpp[i]))*
        (CHAto*nhill(H[i])+CEmcAto*nhill(Emc[i]))-uAto*Ato[i];*/
        vAto[i]=CHhAto*hill(Hh[i])+CDppAto*hill(Dpp[i])+
        CHAto*nhill(H[i])+CEmcAto*nhill(Emc[i])-uAto*Ato[i];
    }
}

double hill(double a)
{
    return (pow(a,n)/(1+pow(a,n)));
}

double nhill(double a)
{
    return (1/(1+pow(a,n)));
}
```

D.2. Simulador 2D

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//ESCRITURA
#define INFO "INFO.txt"
#define FICHOUT "2D-DATA[t,n,Hh,Dpp,H,Ato,Dl,N].txt" /*Dirección del fichero de
salida de datos*/
#define PROGRESO //Activar para ver como avanza el programa
#define res 1. /*Marca la resolución (en porcentaje) con la que informa el
programa de su progreso*/
/*#define DEBUG_MATRIX Saca información de parámetros y posibles errores en la
construcción de las matrices por pantalla*/

//PARÁMETROS DE LA RED
#define forma 0 /*Define la forma de la red hexagonal, es decir, marca el
contorno*/
/*
    0 --> Rectangular (Lx,Ly)
*/
#define cc 0//Condiciones de contorno
/*
    0 --> Limitada en eje x (dirección de propagación de la onda), periódica en
    eje y (perpendicular)
*/
#define NactAto 0//Notch activa Ato de forma directa?
/*
    0 --> No (Los coefs de represeión con retardo son los X64)
    1 --> Si (Se diferencian coeficientes de activación, ahora X64, de los de
    represión con retardo, X64r)
*/
#define Lx 60 //Referente al tamaño de la red x6
#define Ly 3 /*Para simular un patron Delta-Notch con cc=0 debe ser múltiplo de
3, x9*/

//CONDICIONES INICIALES
#define ci 0
/*
    0 --> 1/4 de la red ya evolucionada (con concentraciones simples)
    1 --> '0' pero con ruido gaussiano multiplicativo en las concentraciones
    2 --> '0' con patrón Delta-Notch formado
    3 --> Patrón inicial de Ato según 'Lubensky et al.' en la primera columna
    4 --> '2' con patrón de Ato tipo 'Lubensky et al.' en la parte posterior
    5 --> Todo uniforme con C0 menos Hh y Ato excitadas en las células
    correspondientes
*/

#define ruido 0.9 //Máxima amplitud del ruido
```

```
//Concentraciones iniciales (parte anterior)
#define C10 0.1 //CHh0
#define C20 0 //CDpp0
#define C30 0 //CH0
#define C40 0.5 //CAto0
#define C50 0 //CDl0
#define C60 0 //CN0

//Diferencia posterior-anterior (misma referencia numérica)
#define d1 0.6
#define d2 0
#define d3 0
#define d4 0.5
#define d5 0
#define d6 0
#define d55 0 //Aumento de Delta en células 'Delta'
#define d56 0 //Aumento de Delta en células 'Notch'
#define d65 0 //Aumento de Notch en células 'Delta'
#define d66 0 //Aumento de Notch en células 'Notch'*/

//PARÁMETROS DE LA SIMULACIÓN
#define dt 0.001 //Paso temporal
#define ts 30. //Tiempo de simulación
#define rdi 100 /*Relación de datos calculados frente a escritos (se escribe 1
por cada 'rdi' calculados)*/

#define trN 0.01 //Tiempo de retardo en el efecto de Notch sobre Ato
#define trHh 1 //Tiempo de retardo en el efecto de Hh sobre Ato
#define rHh (int(trHh/dt)) //Número de pasos de retardo en Hh
#define rN (int(trN/dt)) //Número de pasos de retardo en N

//PARÁMETROS DEL SISTEMA
#define Np 6 //Número de parámetros considerados

#define exph 3 //Coeficiente de hill de las funciones

//Coef. de difusión (D/u1)
#define D1 0.8
#define D2 1.5

//Coeficientes de degradación (uX/uHh)
#define u2 1
#define u3 1
#define u4 1
#define u5 1
#define u6 1

//Coeficientes de interacción (C=H/(u1.Kref))
#define H11 0 //Hxy --> coef (adimensionalizado) de x sobre y
#define H12 2
```

```
#define H13 0
#define H14 4
#define H15 0
#define H23 0
#define H24 0.5
#define H25 0
#define H34 0
#define H41 1.27
#define H53 0
#define H56 0//No debería existir
#define H56nn 0//100 //Interacción Delta -> Notch a primeros vecinos
#define H63 0
#define H64 0 /*Interacción N -> Ato directa (Si N->Ato, si no será la de
retardo N-|Ato)*/
#define H65 0
#define H14r 1.1 //Interacción Hh -| Ato con retardo
#define H64r 0 //Interacción Notch -| Atonal con retardo (Si N->Ato)

//Coeficientes de Umbral
#define K23 0.2
#define K24 0.2
#define K25 2
#define K34 1
#define K53 1
#define K56 1//Inutil
#define K63 1
#define K64 1
#define K65 20
#define K64r 1 //Interacción Notch -> Atonal con retardo
#define K14r 0.1

//Valores necesarios, no manipular
#define NormRANu (2.3283063671E-10F) /*Nos va a servir para normalizar los
números aleatorios*/
#define PI 3.14159265359

//Para Box-Muller

unsigned int rueda[256]; //Donde vamos almacenar los aleatorios
unsigned char ind_rand, ig1, ig2, ig3; //Variables auxiliares
float r1, r2; //Número aleatorios generado entre 0 y 1

//VARIABLES DECLARADAS FUERA POR CONVENIENCIA
bool AI[Np][Np];
int nn[Lx*Ly][Lx*Ly];
/*
    AI[i][j] --> Coef. 0 si 'j' es inhibidor de 'i' y 1 si es activador.
    nn[i][j] --> Matriz que marca los primeros vecinos de cada coordenada.
    (¿Pesada en caso de frontera reflectante?)
*/
double K[Np][Np],H[Np][Np],D[Np],U[Np];
```

```
/*DATOS
    K[i][j] --> Parámetro de 'trigger' de la funcion de hill dependiente del
    prámetro 'j' en 'vi'
    H[i][j] --> Coeficiente que acompaña a las funciones de hill con el mismo
    criterio que antes
    D[i] -----> Coeficiente de difusión del componente 'i'
    U[i] -----> Coeficiente de degradación del componente 'i'
*/
FILE *fout,*info;
/*
    *fout --> Fichero de salida de datos
*/

//FUNCIONES
void iniciaRan(int Semilla);
float Parisi_Rapuano(void);
void Box_Muller(double *g1, double *g2);

double h(double C, double k, bool ai);//Función de Hill

void setAI();//Construye la matriz de relaciones activación-inhibición
void setK();//Construye K
void setH();//Construye la matriz de coeficientes de hill
void setD();//Construye el vector 'Coeficientes de difusión'
void setU();//Construye el vector 'Coeficientes de degradación'
void setnn();//Construye la matriz de conectividad (espacial)

void CI(double C[Lx*Ly][Np], double CNr[Lx*Ly][rN],
double CHhr[Lx*Ly][rHh]);/*Aplica las condiciones iniciales a la matriz de
concentraciones*/
void Laplaciano(double C[Lx*Ly][Np], double Lap[Lx*Ly][Np]);
//Calcula el Laplaciano de las diferentes concentraciones
void Velocidades(double C[Lx*Ly][Np], double Vel[Lx*Ly][Np],
double CNr[Lx*Ly][rN], double CHhr[Lx*Ly][rHh]);/* Calcula el cambio en
concentración de cada componente en cualquier punto del espacio*/
void Evolucion(double C[Lx*Ly][Np], double CNr[Lx*Ly][rN],
double CHhr[Lx*Ly][rHh]);//Avanza el sistema un tiempo dt
void Registra(double C[Lx*Ly][Np], double t);//Guarda en fichero

int main()
{
    iniciaRan(time(NULL));
    int aux,aux2,i,j,k;
    double t,C[Lx*Ly][Np],CNr[Lx*Ly][rN],CHhr[Lx*Ly][rHh];
    /*DATOS
        t --------------> Marca el tiempo durante la simulación
        C[i][j] --------> Concentración del parámetro 'j' en la célula 'i'
        CNr[Lx*Ly][rN] --> Guarda las concentraciones de Notch de tiempos
        previos suficientes
        CHhr[Lx*Ly][rHh] --> Guarda las concentraciones de Hedgehog de tiempos
        previos suficientes
```

```
*/
setAI();
setK();
setH();
setD();
setU();
setnn();
#ifdef PROGRESO
{
    printf("Construidas las matrices de interaccion (AI,K,H,D,U)\n");
}
#endif // PROGRESO
#ifdef DEBUG_MATRIX
{
    printf("\nAI:\n\n");//INFO AI
    for(i=0;i<Np;i++)
    {
        for(j=0;j<Np;j++)
            printf("\t%d",AI[i][j]);
        printf("\n");
    }
    printf("\nK:\n\n");//INFO K
    for(i=0;i<Np;i++)
    {
        for(j=0;j<Np;j++)
            printf("\t%lf",K[i][j]);
        printf("\n");
    }
    printf("\nH:\n\n");//INFO H
    for(i=0;i<Np;i++)
    {
        for(j=0;j<Np;j++)
            printf("\t%lf",H[i][j]);
        printf("\n");
    }
    printf("\nD:\t(");//INFO D
    for(i=0;i<Np;i++)
        printf("%lf\t",D[i]);
    printf(")\n");
    printf("\nU:\t(");//INFO U
    for(i=0;i<Np;i++)
        printf("%lf\t",U[i]);
    printf(")\n");
    printf("\nnn:\n\n");//INFO nn
    for(i=0;i<Lx*Ly;i++)
    {
        for(j=0;j<Lx*Ly;j++)
            printf("\t%d",nn[i][j]);
        printf("\n");
    }
}
```

```
#endif // DEBUG_MATRIX
CI(C,CNr,CHhr);//Imponemos las condiciones inciales
#ifdef PROGRESO
{
    printf("Construida las matriz de concentraciones (C[L*L][Np]) "
    "imponiendo condiciones iniciales.\n");
}
#endif // PROGRESO
#ifdef DEBUG_MATRIX
{
    for(k=0;k<Np;k++)
    {
        printf("\nC(%d):\n",k);
        for(i=0;i<Ly;i++)
        {
            for(j=0;j<Ly;j++)
                printf("\t%lf",C[Ly*j+i][k]);
            printf("\n");
        }
    }
}
#endif // DEBUG_MATRIX
info=fopen(INFO,"w");
if(info==NULL)
{
    printf("Error al abrir el fichero de informacion.\nCIERRE DEL "
    "PROGRAMA");
    return 0;
}
fout=fopen(FICHOUT,"w");
if(fout==NULL)
{
    printf("Error al abrir el fichero de salida.\nCIERRE DEL PROGRAMA");
    return 0;
}
#ifdef PROGRESO
{
    printf("Exito al abrir el fichero de salida ("FICHOUT")\n");
}
#endif // PROGRESO
fprintf(info,"%d    %d  %lf\n",Lx,Ly,ts);
fclose(info);
Registra(C,0);//Guardamos la configuración inicial a tiempo 0
#ifdef PROGRESO
{
    printf("\nCondiciones iniciales registradas, comenzamos la "
    "simulacion:\n\n");
}
#endif // PROGRESO
aux=res;
aux2=0;
```

```
    for(t=0;t<ts;t+=dt)
    {
        aux2++;
        Evolucion(C,CNr,CHhr);
        if(aux2%(int(rdi))==0)
            Registra(C,t+dt);
        #ifdef PROGRESO
        {
            if(100*(t+dt)/ts>=aux)
            {
                printf("\t%d%c\n",int(aux),'%');
                aux+=res;
            }
        }
        #endif // PROGRESO
    }
    fclose(fout);
}

void iniciaRan(int Semilla) //Prepara el programa para ejecutar Parisi_Rapuano
{
    int ini, factor, sum, i; //Variables para inicializar la rueda

    srand(Semilla); /*Inicializamos el generador de C con la semilla (No es
    necesarios, pero por si se usa)*/

    ini = Semilla; //Inicializamos las variables
    factor = 675423; //No estoy seguro de por que estas
    sum = 78114698;

    for(i=0; i<256; i++) //Inicializamos la rueda
    {
        ini = (ini*factor+sum);
        rueda[i] = ini;
    }
    ind_rand = ig1 = ig2 = ig3 = 0; /*Inicializamos las variables auxiliares del
    Parisi-Rapuano*/
}

float Parisi_Rapuano(void)
{
    /*
    Devuelve un numero entre 0 y 1 aleatorio plano siguiendo el método
    Parisi-Rapuano.
    Para que funcione deben estar definidas globalmente unsigned float r,
    int rueda[256] y unsigned
    char ind_rand, ig1, ig2 e ig3. Asi como NormRANu. Además en su primera
    ejecución: ind_rand=ig1=ig2=ig3=0
    y en cada rueda[i] debe haber un numero aleatorio (no importa su calidad).
    */
```

```
    float s;

    ig1 = ind_rand - 23;
    ig2 = ind_rand - 129;
    ig3 = ind_rand - 74;

    rueda[ind_rand] = rueda[ig1] + rueda[ig2];

    s = (rueda[ind_rand]^rueda[ig3])*NormRANu;

    ind_rand++;

    return s;
}

void Box_Muller(double *g1, double *g2)/*Genera dos 'doubles' de distribución
Gaussiana*/
{
    r1 = Parisi_Rapuano();//Generamos aleatorios planos
    r2 = Parisi_Rapuano();
    *g1 = -sqrt(-2*log((double)r1))*cos(2*PI*(double)r2);
    //Les damos distribución Gaussiana
    *g2 = -sqrt(-2*log((double)r1))*sin(2*PI*(double)r2);
}

double h(double C, double k, bool ai)
{
    double aux;
    aux=pow(C/k,exph)/(1+pow(C/k,exph));
    if(ai)
        return aux;
    else
        return 1-aux;
}

void setAI()
{
    int i,j;
    for(i=0;i<Np;i++)//Inicializamos a 0
        for(j=0;j<Np;j++)
            AI[i][j]=0; //Inicialmente X-|Y para todo X,Y
    AI[0][0]=1; //Hh -> Hh
    AI[0][3]=1; //Ato -> Hh
    AI[1][0]=1; //Hh -> Dpp
    AI[2][1]=1; //Dpp -> H
    AI[3][0]=1; //Hh -> Ato
    AI[3][1]=1; //Dpp -> Ato
    if(NactAto)
        AI[3][5]=1; //N -> Ato
    AI[4][0]=1; //Hh -> Dl
    AI[4][1]=1; //Dpp -> Dl
```

```
}

void setK()
{
    int i,j;
    for(i=0;i<Np;i++)
        for(j=0;j<Np;j++)
            K[i][j]=1;
    K[2][1]=K23;
    K[3][1]=K24;
    K[4][1]=K25;
    K[3][2]=K34;
    K[2][4]=K53;
    K[5][4]=K56;
    K[2][5]=K63;
    K[3][5]=K64;
    K[4][5]=K65;
}

void setH()
{
    int i,j;
    for(i=0;i<Np;i++)
        for(j=0;j<Np;j++)
            H[i][j]=0;
    H[0][0]=H11;
    H[1][0]=H12;
    H[2][0]=H13;
    H[3][0]=H14;
    H[4][0]=H15;
    H[2][1]=H23;
    H[3][1]=H24;
    H[4][1]=H25;
    H[3][2]=H34;
    H[0][3]=H41;
    H[2][4]=H53;
    H[5][4]=H56;
    H[2][5]=H63;
    H[3][5]=H64;
    H[4][5]=H65;
}

void setD()
{
    int i;
    for(i=0;i<Np;i++)//Marcamos difusión
        D[i]=0;
    D[0]=D1;
    D[1]=D2;
}
```

```
void setU()
{
    U[0]=1;//Marcamos degradación
    U[1]=u2;
    U[2]=u3;
    U[3]=u4;
    U[4]=u5;
    U[5]=u6;
}

void setnn()
{
    int i,j,A,lx,ly,pv[6];
    /*
        i,j ----> Índices auxiliares
        A ------> Área, dimensión de la matriz conectividad (AxA)
        lx,ly --> Lados del sistema (pasado a entero seguro)
        pv[i] --> Coordenadas de los primeros vecinos a una célula
    */
    lx=int(Lx);
    ly=int(Ly);
    A=lx*ly;
    for(i=0;i<A;i++)
        for(j=0;j<A;j++)
            nn[i][j]=0;//Inicializamos a 0;
    switch(cc)//Adaptamos la conectividad a las condiciones de contorno
    {
    case 0:
        for(i=0;i<ly;i++)//Primera columna
        {
            pv[0]=(i+1)%ly+(i/ly)*ly;//n
            pv[1]=(i-1+ly)%ly+(i/ly)*ly;//s
            pv[2]=(i+(i/ly)%2)%ly+(i/ly)*ly+ly;//ne
            pv[3]=(i-1+ly+(i/ly)%2)%ly+(i/ly)*ly+ly;//se
            for(j=0;j<4;j++)
                nn[i][pv[j]]=1;
        }
        for(i=(ly-1)*lx;i<A;i++)//Última columna
        {
            pv[0]=(i+1)%ly+(i/ly)*ly;//n
            pv[1]=(i-1+ly)%ly+(i/ly)*ly;//s
            pv[2]=(i+(i/ly)%2)%ly+(i/ly)*ly-ly;//no
            pv[3]=(i-1+ly+(i/ly)%2)%ly+(i/ly)*ly-ly;//so
            for(j=0;j<4;j++)
                nn[i][pv[j]]=1;
        }
        for(i=ly;i<A-ly;i++)//Resto de malla
        {
            pv[0]=(i+1)%ly+(i/ly)*ly;//n
            pv[1]=(i-1+ly)%ly+(i/ly)*ly;//s
            pv[2]=(i+(i/ly)%2)%ly+(i/ly)*ly+ly;//ne
```

```
            pv[3]=(i-1+ly+(i/ly)%2)%ly+(i/ly)*ly+ly;//se
            pv[4]=(i+(i/ly)%2)%ly+(i/ly)*ly-ly;//no
            pv[5]=(i-1+ly+(i/ly)%2)%ly+(i/ly)*ly-ly;//so
            for(j=0;j<6;j++)
                nn[i][pv[j]]=1;
        }
    }
}

void CI(double C[Lx*Ly][Np], double CNr[Lx*Ly][rN],
double CHhr[Lx*Ly][rHh])
//Aplica las condiciones iniciales a la matriz de concentraciones
{
    int i,j;
    double err,aux;
    switch(forma)
    {
    case 0:
        for(i=0;i<Lx*Ly;i++)
        {
            C[i][0]=C10;
            C[i][1]=C20;
            C[i][2]=C30;
            C[i][3]=C40;
            C[i][4]=C50;
            C[i][5]=C60;
        }
    break;
    default:
        printf("Forma de la red no concretada correctamente, cierre el "
        "programa\n");
    }
    switch(ci)
    {
    case 0:
        for(i=0;i<(int(Lx)/4)*int(Ly);i++)
        {
            C[i][0]+=d1;
            C[i][1]+=d2;
            C[i][2]+=d3;
            C[i][3]+=d4;
            C[i][4]+=d5;
            C[i][5]+=d6;
        }
    break;
    case 1:
        for(i=0;i<(int(Lx)/4)*int(Ly);i++)
        {
            C[i][0]+=d1;
            C[i][1]+=d2;
            C[i][2]+=d3;
```

```
            C[i][3]+=d4;
            C[i][4]+=d5;
            C[i][5]+=d6;
        }
        for(i=0;i<Lx*Ly;i++)
        {
            for(j=0;j<Np;j++)
            {
                Box_Muller(&err,&aux);
                C[i][j]+=C[i][j]*ruido*err;
                if(C[i][j]<0)
                    C[i][j]=0;
            }
        }
    break;
    case 2:
        for(i=0;i<(int(Lx)/4)*int(Ly);i++)
        {
            C[i][0]+=d1;
            C[i][1]+=d2;
            C[i][2]+=d3;
            C[i][3]+=d4;
            C[i][4]+=d5;
            C[i][5]+=d6;
            if(((i-(i/int(Ly))%2)%3)==0)
            {
                C[i][4]+=d55;
                C[i][5]+=d65;
            }
            else
            {
                C[i][4]+=d56;
                C[i][5]+=d66;
            }
        }
    break;
    case 3://Ly debe ser múltiplo de 9 en c.c. periódicas en 'y'
        for(i=0;i<int(Ly);i++)
            if(i%9==0)
            {
                C[i][3]+=d4;
                C[i][0]+=d1;
                C[i][1]+=d2;
                C[i][2]+=d3;
                C[i][4]+=d55;
                C[i][5]+=d65;
            }
    break;
    case 4:
        for(i=0;i<(int(Lx)/4)*int(Ly);i++)
        {
```

```
            C[i][0]+=d1;
            C[i][1]+=d2;
            C[i][2]+=d3;
            C[i][4]+=d5;
            C[i][5]+=d6;
            if(((i-(i/int(Ly))%2)%3)==0)
            {
                C[i][4]+=d55;
                C[i][5]+=d65;
                if((i/int(Ly))%3==0&&(i-4*((i/(3*int(Ly)))%2))%9==0)
                    C[i][3]+=d4;
            }
            else
            {
                C[i][4]+=d56;
                C[i][5]+=d66;
            }
        }
    break;
    case 5:
        for(i=0;i<int(Lx)*int(Ly);i++)
        {
            if((i/int(Ly))%3==0&&(i-4*((i/(3*int(Ly)))%2))%9==0)
            {
                C[i][0]+=d1;
                C[i][3]+=d4;
            }
        }
    break;
    default:
        printf("Modo de condici%cn inicial no programado (o no reconocido), "
        "fin del programa\n",'ó');
    }
    for(i=0;i<Lx*Ly;i++)
    {
        for(j=0;j<rHh;j++)
            CHhr[i][j]=C[i][0];
        for(j=0;j<rN;j++)
            CNr[i][j]=C[i][5];
    }
}
void Laplaciano(double C[Lx*Ly][Np], double Lap[Lx*Ly][Np])/*Calcula el
Laplaciano de las diferentes concentraciones*/
{
    int i,j,k;
    for(i=0;i<Lx*Ly;i++)//Inicializamos la matriz
        for(j=0;j<Np;j++)
            Lap[i][j]=0;
    for(i=0;i<Lx*Ly;i++)
        for(j=0;j<Np;j++)
            for(k=0;k<Lx*Ly;k++)
```

```
                Lap[i][j]+=double(nn[i][k])*(C[k][j]-C[i][j]);
}

void Velocidades(double C[Lx*Ly][Np], double Vel[Lx*Ly][Np],
double CNr[Lx*Ly][rN], double CHhr[Lx*Ly][rHh])/* Calcula el cambio en
concentración de cada componente en cualquier punto del espacio*/
{
    int i,j,k,SCDnn;
    double Lap[Lx*Ly][Np];//Laplaciano y coeficientes de difusión y degradación
    Laplaciano(C,Lap);//Calculamos el laplaciano de las componentes de interés
    for(i=0;i<Lx*Ly;i++)
    {
        for(j=0;j<Np;j++)//Términos intracelulares y difusión
        {
            Vel[i][j]=D[j]*Lap[i][j]-U[j]*C[i][j];
            for(k=0;k<Np;k++)
            {
                Vel[i][j]+=H[j][k]*h(C[i][k],K[j][k],AI[j][k]);
            }
        }
        k=0;/*Pasamos a usarlo como contador de los primeros vecinos de una
        célula para poder calcular el promedio*/
        SCDnn=0;/*Inicializamos la suma de las concentraciones iniciales de
        Delta en primeros vecinos*/
        for(j=0;j<Lx*Ly;j++)//Términos de ligandos en membranas
            if(nn[i][j]!=0)
            {
                SCDnn+=C[j][4];
                k++;
            }
        SCDnn/=k;
        Vel[i][5]+=H56nn*h(SCDnn,1,1);
        //Notch normalizado a la interacción con Delta en nn, activación.
        if(NactAto)
            Vel[i][3]+=H64r*h(CNr[i][rN-1],K64r,0);//Retardo N-|Ato
        Vel[i][3]+=H14r*h(CHhr[i][rHh-1],K14r,0);//Retardo Hh-|Ato
    }
}

void Evolucion(double C[Lx*Ly][Np], double CNr[Lx*Ly][rN],
double CHhr[Lx*Ly][rHh])//Avanza el sistema un tiempo dt
{
    int i,j;
    double Vel[Lx*Ly][Np];
    Velocidades(C,Vel,CNr,CHhr);
    for(i=0;i<Lx*Ly;i++)
    {
        for(j=0;j<Np;j++)
            C[i][j]+=Vel[i][j]*dt;
        for(j=rN-1;j>0;j--)
            CNr[i][j]=CNr[i][j-1];
```

```
            //Vamos desplazando los datos de retardo guardados
        CNr[i][0]=C[i][5];
        for(j=rHh-1;j>0;j--)
            CHhr[i][j]=CHhr[i][j-1];
        CHhr[i][0]=C[i][0];
    }
}

void Registra(double C[Lx*Ly][Np], double t)
{
    int i;
    for(i=0;i<Lx*Ly;i++)
        fprintf(fout,"%lf   %d  %lf %lf %lf %lf %lf %lf\n", t, i, C[i][0],
        C[i][1], C[i][2], C[i][3], C[i][4], C[i][5]);
}
```

D.3. Análisis de la velocidad del MF

```
//LIBRERIAS
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//---PARÁMETROS DEL PROGRAMA---

//Eltos. a medir
#define VG //Velocidad de la onda (velocidad de grupo)

//Herramientas
#define PROGRESO //Marca como transcurre la simulación.
#define res 10 //Resolución en porcentaje

//SUBALGORITMOS

void Vel_Grup(int Nt, int N, int CI);
//Calcula la velocidad de grupo para cada instante de tiempo

//CÓDIGO PRINCIPAL
int main()
{
    //CARGA DE DATOS
    FILE *in;//Parámetros del programa
    int N,Nt,CI;
    in=fopen("INFO.txt","r");
    fscanf(in,"%d %d %d",&Nt,&N,&CI);

    //ANÁLISIS DE LOS RESULTADOS
    #ifdef VG
    #ifdef PROGRESO
    printf("Comienza el c%clculo de la velocidad de grupo\n\n",'á');
    #endif // PROGRESO
    Vel_Grup(Nt,N,CI);
    #endif // VG
}

//SUBALGORITMOS

void Vel_Grup(int Nt, int N, int CI)
{
    FILE *datos;
    FILE *vel;
    datos=fopen("Datos[t,x,Hh,Dpp,Hairy,Emc,Ato].txt","r");
    vel=fopen("Velocidad de grupo [t,VG].txt","w");
    int i,j,aux1,x,xi,xf;
    double aux,t,vg,vgmed,vgmed2,ti;
    double s[N], smax, smin, smed;
```

```
xi=0;
xf=0;
ti=0;
t=0;
vgmed=0;
vgmed2=0;
if(CI==3)
{
    for(i=0;i<Nt;i++)
    {
        smax=0;
        smin=9999;
        ti=t;
        xi=xf;
        vg=0;
        for(j=0;j<N;j++)
        {
            fscanf(datos,"%lf   %d  %lf %lf %lf %lf %lf",
            &t,&aux1,&aux,&s[j],&aux,&aux,&aux);
            if(s[j]>smax)
                smax=s[j];
            if(s[j]<smin)
                smin=s[j];
        }
        smed=smax/2+smin/2;
        x=N/5;
        while(s[x]>smed)
        {
            x++;
        }
        xf=x;
        vg=(double(xf)-double(xi))/(t-ti);
        if(t==ti)
            vg=0;
        fprintf(vel,"%lf    %lf\n",t,vg);
        #ifdef PROGRESO
        if(((i+1)*100)%(Nt*int(res))==0)
            printf("%d%c\n",(i+1)*100/Nt,'%');
        #endif // PROGRESO
        vgmed+=vg;
        vgmed2+=vg*vg;
    }
    vgmed/=Nt;
    vgmed2/=Nt;
    printf("\nVelocidad media del \"furrow\": %lfñ%lfu.a\n\n",vgmed ,
    sqrt(vgmed2-vgmed*vgmed)/sqrt(Nt));
    fclose(datos);
    fclose(vel);

}
```

```
    else
        printf("Algoritmo no programado para el caso '3', a%cadir al "
        " subalgoritmo 'Vel_Grup'\n",'ñ');
}
```

D.4. Representación 1D

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define GNUPLOT "C:\\ProgramFiles\\gnuplot\\bin\\gnuplot.exe -persistent"
//Direccion de gnuplot manteniendo la ventana abierta
#define INFO "INFO.txt" //Nombre del fichero con los datos de N y Nt
#define DATOS "Datos[t,x,Hh,Dpp,Hairy,Emc,Ato].txt"
//Nombre del fichero con los datos a representar

#define cMax 7.0 //Concentración máxima a representar
#define delay 30
#define pausa 0.3
//Tiempo en segundos que se le da a gnuplot para procesar la información

void sleep(double tiempo);

int main()
{
    int i,j,Nt,N,aux,x;
    /*DATOS:
        i,j,aux --> Variables auxiliares
        Nt -------> Número de pasos temporales registrados en el fichero de datos a repre
        N --------> Número de células 'x' involucradas en el proceso
        x --------> Posición
    */
    FILE *gp,*info,*datos,*faux;
    /*
        gp -----> 'Pipe' a Gnuplot
        info ---> Fichero con la información de Nt y N
        datos --> Fichero con los datos a tratar
        faux ---> Fichero auxiliar de representación
    */
    double t,Hh,Dpp,H,Emc,Ato;//Concentracion de genes/proteinas
    printf("Ruta a Gnuplot utilizada: ");
    printf(GNUPLOT);
    printf("\n");
    gp=popen(GNUPLOT,"w");//Llamamos a Gnuplot
    if(gp==NULL)
    {
        printf("Error al abrir la ruta a gnuplot.\nCese del programa");
        return 0;
    }
    fprintf(gp,"plot 'Datos(fin)[t,x,Hh,Dpp,Hairy,Emc,Ato].txt' u 2:3 w l t "
    "'Hh'\n");
    fflush(gp);
    fprintf(gp,"set out\n");
    printf("Abriendo el fichero ");
    printf(INFO);
```

```
printf("\n");
info=fopen(INFO,"r");//Abrimos el fichero INFO
if(info==NULL)
{
    printf("Error al abrir el fichero ");
    printf(INFO);
    printf("\nCese del programa");
    return 0;
}
printf("Abriendo el fichero ");
printf(DATOS);
printf("\n");
datos=fopen(DATOS,"r");//Abrimos el fichero DATOS
if(datos==NULL)
{
    printf("Error al abrir el fichero ");
    printf(DATOS);
    printf("\nCese del programa");
    return 0;
}
fscanf(info,"%d %d  %d",&Nt,&N,&aux);//Leemos el fichero INFO
//CONFIGURAMOS GNUPLOT
//fprintf(gp,"cd 'C:\\Users\\ruber\\OneDrive - unizar.es\\4%c\\TFG'\n",'º');
fprintf(gp,"cd 'C:\\Users\\Alejandro\\OneDrive - unizar.es\\4%c\\TFG'\n",'º');

//fprintf(gp,"set xrange[%d/5:%d/2]\n",N,N);
//fprintf(gp,"set xtics %d font \",40\"\n",N/4);
fprintf(gp,"unset xtics\n");
fprintf(gp,"set ytics %d font \",40\"\n",6);
fprintf(gp,"set xrange[0:%d]\n",N);
fprintf(gp,"set yrange[0:%lf]\n",cMax);
//fprintf(gp,"set xlabel \'Posici%cn (c%clula)\' font \",40\"\n",'ó','é');
//fprintf(gp,"set ylabel \'Concentraci%cn (adimensionalizada)\' "
"font \",40\"\n",'ó');
//MONTAMOS EL GIF
printf("Comienza a construirse el GIF\n");
char a;
for(i=0;i<Nt;i++)
{
    faux=fopen("Auxiliar.txt","w");
    for(j=0;j<N;j++)
    {
        fscanf(datos,"%lf   %d  %lf %lf %lf %lf %lf",
        &t,&x,&Hh,&Dpp,&H,&Emc,&Ato);
        fprintf(faux,"%d    %lf %lf %lf %lf %lf\n",x,Hh,Dpp,H,Emc,Ato);
    }
    fclose(faux);
    /*fprintf(gp,"set title \'t=%lf\'\n",t);
    fprintf(gp,"plot 'Auxiliar.txt' u 1:2 w l lw 3 t 'Hh'\n");
    fprintf(gp,"replot \'Auxiliar.txt\' u 1:4 w l lw 3 t \'Hairy\'\n");
    //fprintf(gp,"replot \'Auxiliar.txt\' u 1:3 w l lw 3 t \'Dpp\'\n");
```

```
        //fprintf(gp,"replot \'Auxiliar.txt\' u 1:5 w l lw 3 t \'Emc\'\n");
        fprintf(gp,"replot \'Auxiliar.txt\' u 1:6 w l lw 3 lc rgb 'red' t "
        "\'Ato\'\n");*/
        fprintf(gp,"plot \'Auxiliar.txt\' u 1:2 w l lw 3 t \'\'\n");
        fprintf(gp,"replot \'Auxiliar.txt\' u 1:4 w l lw 3 t \'\'\n");
        fprintf(gp,"replot \'Auxiliar.txt\' u 1:3 w l lw 3 t \'\'\n");
        //fprintf(gp,"replot \'Auxiliar.txt\' u 1:6 w l lw 3 lc rgb \'red\' "
        "t \'\'\n");
        fflush(gp);
        if((100*(i+1))%Nt==0)
            printf("%d%c\n",100*(i+1)/Nt,'%');
        sleep(pausa);
    }
    //fflush(gp);
    pclose(gp);//Cerramos el 'pipe' a Gnuplot
    return 0;
}

void sleep(double tiempo)
{
    int i;
    double a;
    for(i=0;i<int(tiempo*3450000);i++)
        a=exp(a*i);
}
```

D.5. Representación 2D

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

//LECTURA-ESCRITURA
//#define INFO "INFO_3.txt"
//#define DATA "Data_3.txt"

#define INFO "INFO.txt"
//#define INFO "INFO_OscConDiff.txt"
//#define INFO "Info_DlNPerf.txt"
//#define INFO "INFO_HA.txt"
#define DATA "2D-DATA[t,n,Hh,Dpp,H,Ato,Dl,N].txt"
//#define DATA "Data_OscConDiff.txt"
//#define DATA "Data_DlNPerf.txt"
//#define DATA "Data_HA.txt"
#define GNUPLOT "C:\\ProgramFiles\\gnuplot\\bin\\gnuplot.exe -persistent"
//Direccion de gnuplot manteniendo la ventana abierta

//PARÁMETROS DEL GIF
#define ndpp 10//Número de datos por plot
#define delay 10
#define pausa 0.1

//SELECCIÓN DE CONCENTRACIONES
//(1 si queremos hacer el gif en esa concentración, 0 si no)
#define Hh 0
#define Dpp 0
#define H 0
#define Ato 1
#define Dl 0
#define N 0

#define mode 0// Si queremos representar simultáneamente las concentraciones
/*
    0 --> Gifs de cada concentración consecutivos (una única ventana)
    1 --> Preparado la representar en una misma ventana simultáneamente Hh,Dl y
    N.
    2 --> 2 Ventanas, una superior con el MF (Hh,Dpp,H y Ato a elección), y una
    inferior con la interfaz Dl-N
*/

void sleep(double tiempo);
void BuscaMaxMin(double Cmin[6], double Cmax[6]);

void Caso0(int Lx, int Ly, int L, double Cmin[6], double Cmax[6]);
void Caso1(int Lx, int Ly, int L, double Cmin[6], double Cmax[6]);
void Caso2(int Lx, int Ly, int L, double Cmin[6], double Cmax[6]);
```

```
FILE *gp,*data;
bool P[6];

int main()
{
    int i,j,k,p,Lx,Ly,L,aux;
    P[0]=Hh;
    P[1]=Dpp;
    P[2]=H;
    P[3]=Ato;
    P[4]=Dl;
    P[5]=N;
    double ts,t,A,R,G,B,Cmin[6],Cmax[6];
    FILE *info;

    info=fopen(INFO,"r");
    if(info==NULL)
    {
        printf("Error al abrir el fichero "INFO".\nCese del programa");
        return 0;
    }
    fscanf(info,"%d %d  %lf",&Lx,&Ly,&ts);
    fclose(info);

    double C[Lx*Ly][6];

    BuscaMaxMin(Cmin,Cmax);
    data=fopen(DATA,"r");
    if(data==NULL)
    {
        printf("Error al abrir el fichero "DATA".\nCese del programa");
        return 0;
    }

    printf("Ruta a Gnuplot utilizada: "GNUPLOT"\n");
    gp=popen(GNUPLOT,"w");
    if(gp==NULL)
    {
        printf("Error al abrir la ruta a gnuplot.\nCese del programa");
        return 0;
    }
    fprintf(gp,"set terminal wxt transparent enhanced font \"Verdana,22\" "
    "size 2000,2000\n");
    //fprintf(gp,"unset terminal size 1500,1500\n");

    L=Lx;
    if(L<Ly)
        L=Ly;
    switch(mode)
    {
```

```
    case 0:
        Caso0(Lx,Ly,L,Cmin,Cmax);
    break;
    case 1:
        Caso1(Lx,Ly,L,Cmin,Cmax);
    break;
    case 2:
        Caso2(Lx,Ly,L,Cmin,Cmax);
    break;
    default:
        printf("Modo de representación no concretado correctamente, ejecutamos "
        "por defecto el modo 2\n");
        Caso2(Lx,Ly,L,Cmin,Cmax);
    break;
    }
    fclose(gp);
    fclose(data);
}

void sleep(double tiempo)
{
    int i;
    double a;
    for(i=0;i<int(tiempo*3450000);i++)
        a=exp(a*i);
}

void BuscaMaxMin(double Cmin[6], double Cmax[6])
{
    FILE *f;
    double C[6],aux;
    int i,aux1;
    f=fopen(DATA,"r");
    if(f==NULL)
        printf("Error al abrir el fichero "DATA".\nCierre del programa");
    fscanf(f,"%lf    %d  %lf %lf %lf %lf %lf %lf",
    &aux,&aux1,&C[0],&C[1],&C[2],&C[3],&C[4],&C[5]);
    for(i=0;i<6;i++)
    {
        Cmin[i]=C[i];
        Cmax[i]=C[i];
    }
    while(!feof(f))
    {
        fscanf(f,"%lf    %d  %lf %lf %lf %lf %lf %lf",
        &aux,&aux1,&C[0],&C[1],&C[2],&C[3],&C[4],&C[5]);
        for(i=0;i<6;i++)
        {
            if(C[i]<Cmin[i])
                Cmin[i]=C[i];
            if(C[i]>Cmax[i])
```

```
                Cmax[i]=C[i];
            }
        }
        fclose(f);
}

void Caso0(int Lx, int Ly, int L, double Cmin[6], double Cmax[6])
//Representacion de gifs de cada concentración independiente de forma consecutiva
{
    int i,j,k,p;
    double R,G,B,A,t;
    double C[Lx*Ly][6];
    for(i=0;i<6;i++)
    {
        printf("Creando GIF: %d\n",i);
        //fprintf(gp,"set output \"GIF%d.gif\"\n",i);
        if(double(Ly)>double(Lx)*sqrt(3)/2/1.7)
        {
            fprintf(gp,"set xrange[%d:%lf]\n",0,double(L+1)*1.7);
            fprintf(gp,"set yrange[%d:%d]\n",0,L+1);
        }
        else
        {
            fprintf(gp,"set xrange[%d:%lf]\n",0,double(L+1)*sqrt(3)/2);
            fprintf(gp,"set yrange[%d:%lf]\n",0,double(L+1)/1.7*sqrt(3)/2);
        }
        if(P[i])
        {
            fclose(data);
            data=fopen(DATA,"r");
            j=0;
            if(i<4)
            {
                R=double(i*(i-1)*(i-2)/6)/2;
                G=double(-i*(i-1)*(i-3)/2)/2;
                B=double(i*(i-2)*(i-3)/2)/2;
            }
            else
            {
                G=0;
                R=double(5-i)/2;
                B=double(i-4)/2;
            }
            while(!feof(data))
            {
                for(k=0;k<Lx*Ly;k++)
                {
                    fscanf(data,"%lf    %d  %lf %lf %lf %lf %lf %lf",
                    &t,&p,&C[k][0],&C[k][1],&C[k][2],&C[k][3],&C[k][4],&C[k][5]);
                    A=double((Cmax[i]-C[k][i])/(Cmax[i]-Cmin[i]));
                    if(j%int(ndpp)==0)
```

```
                        fprintf(gp,"set object %d circle at %lf, %lf size %lf "
                        "fc rgb \"#%X\" fs solid 1 linewidth .1\n",
                        k+1,double(k/Ly)*sqrt(3)/2+0.5,
                        double(k%Ly)+0.5*double((k/Ly)%2)+0.5,0.5,
                        ((int(A*255)*256+int(R*255))*256+int(G*255))*256+
                        int(B*255));
                    //sleep(pausa);
                }
                if(j%int(ndpp)==0)
                {
                    switch(i)
                    {
                    case 0:
                        fprintf(gp,"plot 1/0 t 'Hh(t=%lf)'\n",t);
                    break;
                    case 1:
                        fprintf(gp,"plot 1/0 t 'Dpp(t=%lf)'\n",t);
                    break;
                    case 2:
                        fprintf(gp,"plot 1/0 t 'H(t=%lf)'\n",t);
                    break;
                    case 3:
                        fprintf(gp,"plot 1/0 t 'Ato(t=%lf)'\n",t);
                    break;
                    case 4:
                        fprintf(gp,"plot 1/0 t 'Dl(t=%lf)'\n",t);
                    break;
                    case 5:
                        fprintf(gp,"plot 1/0 t 'N(t=%lf)'\n",t);
                    break;
                    default:
                        printf("ERROR(indice no correspondiente a ninguna "
                        "concentracion)\n");
                    break;
                    }
                    //fprintf(gp,"plot 1/0 t ''\n");
                    //sleep(pausa);
                    //fflush(gp);
                    //sleep(pausa);
                    //sleep(pausa);
                }
                j++;
            }
        }
    }
}

void Caso1(int Lx, int Ly, int L, double Cmin[6],
double Cmax[6])//Representación simultánea de Hh,Dl y N
{
    int i,j,k,p,aux;
```

```
double A,R,G,B,t;
double C[Lx*Ly][6];
j=0;
if(Ly>double(Lx)*sqrt(3)/2*1.7)
{
    fprintf(gp,"set xrange[%d:%lf]\n",0,double(L+1)*1.7);
    fprintf(gp,"set yrange[%d:%d]\n",0,L+1);
}
else
{
    fprintf(gp,"set xrange[%d:%lf]\n",0,double(L+1)*sqrt(3)/2);
    fprintf(gp,"set yrange[%d:%lf]\n",0,double(L+1)/1.7*sqrt(3)/2);
}
while(!feof(data))
{
    for(k=0;k<Lx*Ly;k++)
    {
        fscanf(data,"%lf    %d  %lf %lf %lf %lf %lf %lf",
        &t,&p,&C[k][0],&C[k][1],&C[k][2],&C[k][3],&C[k][4],&C[k][5]);
        A=0;
        aux=0;
        for(i=0;i<6;i++)
        {
            A+=P[i]*double((Cmax[i]-C[k][i])/(Cmax[i]-Cmin[i]));
            aux+=P[i];
        }
        A/=aux;
        R=double((C[k][4]-Cmin[4])/(Cmax[4]-Cmin[4]));
        G=0;
        /*G=double((C[k][0]-Cmin[0])/(Cmax[0]-Cmin[0]));<--- Activar en caso
        de querer ver Hh*/
        B=double((C[k][5]-Cmin[5])/(Cmax[5]-Cmin[5]));
        if(j%int(ndpp)==0)
            fprintf(gp,"set object %d circle at %lf, %lf size %lf fc "
            "rgb \"#%X\" fs solid 1 linewidth .1\n",
            k+1,double(k/Ly)*sqrt(3)/2+0.5,
            double(k%Ly)+0.5*double((k/Ly)%2)+0.5,0.5,
            ((int(A*255)*256+int(R*255))*256+int(G*255))*256+int(B*255));
        //sleep(pausa);
    }
    if(j%int(ndpp)==0)
    {
        fprintf(gp,"plot 1/0 t 't=%lf'\n",t);
        //fprintf(gp,"plot 1/0 t ''\n");
        //sleep(pausa);
        //fflush(gp);
        //sleep(pausa);
        //sleep(pausa);
    }
    j++;
}
```

```
}

void Caso2(int Lx, int Ly, int L, double Cmin[6], double Cmax[6])
/*Representación simultánea de todas las concentraciones en dos ventanas
(una para el MF y otra para Dl-N)*/
{
    int i,j,k,aux,p;
    double C[Lx*Ly][6];
    double A1,A2,R1,R2,G1,G2,B1,B2,t;
    j=0;
    if(2*Ly>double(Lx)*sqrt(3)/2*1.7)
    {
        fprintf(gp,"set xrange[%d:%lf]\n",0,2*double(L+1)*1.7);
        fprintf(gp,"set yrange[%d:%d]\n",-L-1,L+1);
    }
    else
    {
        fprintf(gp,"set xrange[%d:%lf]\n",0,2*double(L+1)*sqrt(3)/2);
        fprintf(gp,"set yrange[%lf:%lf]\n",-double(L+1)/1.7*sqrt(3)/2,
        double(L+1)/1.7*sqrt(3)/2);
    }
    while(!feof(data))
    {
        for(k=0;k<Lx*Ly;k++)
        {
            A1=0;
            A2=0;
            aux=0;
            fscanf(data,"%lf    %d  %lf %lf %lf %lf %lf %lf",
            &t,&p,&C[k][0],&C[k][1],&C[k][2],&C[k][3],&C[k][4],&C[k][5]);
            for(i=0;i<4;i++)
            {
                A1+=(Cmax[i]-C[k][i])/(Cmax[i]-Cmin[i]);
                aux++;
            }
            A1/=aux;
            aux=0;
            R1=((C[k][3]-Cmin[3])/(Cmax[3]-Cmin[3])*3+
            (C[k][0]-Cmin[0])/(Cmax[0]-Cmin[0]))/4;
            G1=((C[k][2]-Cmin[2])/(Cmax[2]-Cmin[2])*3+
            (C[k][0]-Cmin[0])/(Cmax[0]-Cmin[0]))/4;
            B1=((C[k][1]-Cmin[1])/(Cmax[1]-Cmin[1])*3+
            (C[k][0]-Cmin[0])/(Cmax[0]-Cmin[0]))/4;
            for(i=4;i<6;i++)
            {
                A2+=(Cmax[i]-C[k][i])/(Cmax[i]-Cmin[i]);
                aux++;
            }
            A2/=aux;
            aux=0;
            R2=(C[k][4]-Cmin[4])/(Cmax[4]-Cmin[4]);
```

```
            G2=0;
            B2=(C[k][5]-Cmin[5])/(Cmax[5]-Cmin[5]);
            if(j%int(ndpp)==0)
            {
                fprintf(gp,"set object %d circle at %lf, %lf size %lf fc "
                "rgb \"#%X\" fs solid 1 linewidth .1\n",2*k+1,
                double(k/Ly)*sqrt(3)/2+0.5,
                double(k%Ly)+0.5*double((k/Ly)%2)+0.5,0.5,
                ((int(A1*255)*256+int(R1*255))*256+int(G1*255))*256+int(B1*255));
                fprintf(gp,"set object %d circle at %lf, %lf size %lf fc "
                "rgb \"#%X\" fs solid 1 linewidth .1\n",2*k+2,
                double(k/Ly)*sqrt(3)/2+0.5,
                double(k%Ly)+0.5*double((k/Ly)%2)+0.5-double(Ly+1),0.5,
                ((int(A2*255)*256+int(R2*255))*256+int(G2*255))*256+int(B2*255));
                //sleep(pausa);
            }
        }
        if(j%int(ndpp)==0)
        {
            fprintf(gp,"plot 1/0 t 't=%lf'\n",t);
            //fprintf(gp,"plot 1/0 t ''\n");
            //sleep(pausa);
            //fflush(gp);
            //sleep(pausa);
            //sleep(pausa);
        }
        j++;
    }
}
```