



UNIVERSITY OF
LIVERPOOL

UNIVERSITY OF LIVERPOOL

Modelling Biological Pattern Formation

Alexander Phillips

Supervisor: Dr. Bakhtier Vasiev

Submitted to the Department of Mathematical Sciences in partial fulfilment of the
requirements for the degree of Master of Mathematics

May 25, 2015

Contents

1	Introduction	3
2	Models of Simple Catalytic Reactions	5
2.1	Michaelis-Menten Kinetics	5
2.1.1	Theory	5
2.1.2	Phase-Plane Analysis	7
2.1.3	Quasi-Steady-State Hypothesis	10
2.2	Cooperative Dynamics	11
2.3	Other Types of Dynamical Systems	12
2.4	Distributed Systems	14
3	Diffusion-Decay Systems	15
3.1	Solutions for Diffusion-Decay Systems	15
3.1.1	Dirichlet Boundary Conditions	16
3.1.2	Neumann Boundary Conditions	17
3.1.3	Mixed Boundary Conditions	17
3.2	Scaling in Diffusion-Decay Systems	18
3.2.1	Defining Scaling Factor	18
3.2.2	Comparing Scaling Factors for Neumann and Dirichlet Boundary Conditions	20
3.2.3	Effect of Medium Size on Scaling Factor under Dirichlet Boundary conditions	21
4	Non-linear Reaction-Diffusion Systems	23
4.1	Models	23
4.2	Fitzhugh-Nagumo	24
4.2.1	Travelling Wave	24
4.2.2	Pulsating Spot	24
4.2.3	Stationary Spot	25
4.3	Stability Analysis of the Stationary Spot	26
4.4	Fitzhugh-Nagumo with noise	31
4.5	Radially-Symmetric 2D	32
4.6	Fitzhugh-Nagumo in 2-dimensions	34

4.7 Noise in the 2D FitzHugh-Nagumo System	37
5 Conclusion	39
Appendices	41

Chapter 1

Introduction

Patterns appear everywhere in nature. Examples include the patterns of spots and stripes of pigmentation on the coats of mammals or spotted and striped patterns on the surface of mollusc shells. The shell of a molluscs effectively functions as a 1-dimensional against time plot due to the manner in which the shell grows over time (Meinhardt 2009). The iridescent patterns on the feathers of birds such as the peacock and the wings of butterflies are not caused by pigmentation, but by microscopic differences in the structure of the cells such that the wing surface or feather reflects and diffracts visible light such that it appears to be coloured, rather than it being coloured by a pigment. Despite possible differences in the nature of the colouration, it still requires there to be some underlying structure.

The arrangement of parts of a plant, for instance, leaves on a stem or scales on a cone form a spiral pattern. It was known for many years that these spirals are of the form of Fibonacci spirals. (Edelstein-Keshet 1988, p.497).

In order for these patterns to appear, there must be some underlying mechanism. Turing's seminal paper "The Chemical Basis of Morphogenesis" introduces the general concept of a morphogen. Morphogens are any of a number of different chemicals which influence the differentiation of cells within a developing organism, giving shape to an otherwise formless mass of tissue. The word morphogen itself is derived from the Greek "morphe", meaning shape, and "genesis", meaning beginning or origin: thus, form-giving. These may be promoting the development of different tissue types, organs or skin pigments. For instance, Turing gives a hypothetical example of a "leg-evocator" morphogen promoting the development of a leg organ in the parts of an organism where it is prevalent (Turing 1952, p.39).

For many years, morphogens were largely hypothetical; no specific examples were found. However, in recent times, specific examples have been found: Murray gives the example of calcium in the marine algae *Acetabularia ryukyuenesis* as a morphogen, the distribution of which is directly correlated with the spatial arrangement of hairs in the cap (Murray 1989, p.468). The gene expression found in species such as hydra gives some evidence of these morphogens. In particular the gradient of the organizer; when

the organism is dissected, the two parts regrow head sections in an orientation identical to the original, in correspondence with the preexisting gradient. (Meinhardt 2008, p.15).

In order to properly understand the mechanics which allow morphogens to function in the way that they do, in that they presumably catalyse reactions in order to promote certain characteristics, it is helpful to understand the dynamics of a basic biochemical reaction by studying Michael-Menten Kinetics. This will be discussed in chapter 2.

Furthermore, since the morphogens must somehow propagate through the organism, we consider the implications of diffusion on the scaling of concentration profiles, which will be investigated in chapter 3.

Finally, in chapter 4, we consider models of reaction-diffusion type, in particular the Fitzhugh-Nagumo model, and simulate systems using this model in multiple dimensions in order to observe some simple patterns.

Chapter 2

Models of Simple Catalytic Reactions

2.1 Michaelis-Menten Kinetics

2.1.1 Theory

We consider a relatively simple biochemical reaction wherein an enzyme E binds with molecules of a substrate, S , to form a complex, C . The complex then produces a molecule of the product, P , leaving the original enzyme.



Where k_1 , k_{-1} and k_2 are constant coefficients related to the rate of reaction.

The complex can sometimes degrade back into its constituent parts, the enzyme and substrate. This occurs at a much slower rate than the conversion to the product, so overall, there is a tendency towards the product being produced and the substrate being depleted. Hence $k_{-1} < k_2$.

In order for a pair of molecules to react with each other, they must collide (in some cases, collide in the correct orientation), so as the concentrations increase, the frequency/probability at which the molecules collide in the correct orientation in order to react increases since there are more of them so there are more collisions. So therefore the rate of reaction increases. This is the Law of Mass Action which dictates that the rates of reaction are proportional to the concentration of the reactants.

We denote the concentration of S , E , C and P as s , e , c and p respectively, so we can derive the following system of equations:

$$\frac{ds}{dt} = -k_1es + k_{-1}c \quad (2.2)$$

$$\frac{dc}{dt} = k_1es - (k_{-1} + k_2)c \quad (2.3)$$

$$\frac{de}{dt} = -k_1es + (k_{-1} + k_2)c \quad (2.4)$$

$$\frac{dp}{dt} = k_2c \quad (2.5)$$

Adding (2.3) and (2.4) we get:

$$\frac{de}{dt} + \frac{dc}{dt} = 0 \quad (2.6)$$

$$\Rightarrow e + c = e_0 \quad (2.7)$$

With e_0 being constant. Since enzymes are not created or destroyed during this reaction, it is clear that e_0 is the total number of enzymes.

Then, substituting $e = e_0 - c$ into (2.2) and (2.3):

$$\frac{ds}{dt} = -k_1(e_0 - c)s + k_{-1}c = -k_1e_0s + (k_1s + k_{-1})c \quad (2.8)$$

$$\frac{dc}{dt} = k_1(e_0 - c)s - (k_{-1} + k_2)c = k_1e_0s - (k_1s + k_{-1} + k_2)c \quad (2.9)$$

Non-dimensionalisation allows us to reduce the number of variables.

We non-dimensionalise with:

$$\tau = tk_1e_0 \quad (2.10)$$

$$u = \frac{s}{s_0} \quad (2.11)$$

$$v = \frac{c}{e_0} \quad (2.12)$$

$$\epsilon = \frac{e_0}{s_0} \quad (2.13)$$

Generally, the concentration of enzymes is relatively tiny compared to the concentration of the substrate, so $e_0 \ll s_0$ which means that $\epsilon \ll 1$.

To obtain:

$$\frac{ds}{d\tau} \frac{d\tau}{dt} = k_1e_0 \frac{ds}{d\tau} = -k_1e_0s + (k_1s + k_{-1})c \quad (2.14)$$

$$k_1e_0 \frac{dc}{d\tau} = k_1e_0s - (k_1s + k_{-1} + k_2)c \quad (2.15)$$

Then with $ds = s_0du$, $dc = e_0dv$

$$k_1 e_0 s_0 \frac{du}{d\tau} = k_1 e_0 s_0 u + (k_1 s_0 u + k_{-1}) e_0 v \quad (2.16)$$

$$k_1 e_0^2 \frac{dv}{d\tau} = k_1 e_0 s_0 u - (k_1 s_0 u + k_{-1} + k_2) e_0 v \quad (2.17)$$

Dividing both by $k_1 e_0 s_0$:

$$\frac{du}{d\tau} = -u + \left(u + \frac{k_{-1}}{k_1 s_0}\right)v \quad (2.18)$$

$$\epsilon \frac{dv}{d\tau} = u - \left(u + \frac{k_{-1} + k_2}{k_1 s_0}\right)v \quad (2.19)$$

From here it is convenient to define $\lambda = \frac{k_2}{k_1 s_0}$ and $K = \frac{k_{-1} + k_2}{k_1 s_0}$. Hence equations 2.18 become:

$$\frac{du}{d\tau} = -u + (u + K - \lambda)v \quad (2.20)$$

$$\epsilon \frac{dv}{d\tau} = u - (u + K)v \quad (2.21)$$

From here there are two approaches for analysis.

2.1.2 Phase-Plane Analysis

Null-clines for the u-v plane are:

$$\frac{du}{d\tau} = 0 \Leftrightarrow v = \frac{u}{u + K - \lambda} \quad (2.22)$$

$$\frac{dv}{d\tau} = 0 \Leftrightarrow v = \frac{u}{u + K} \quad (2.23)$$

We consider different behaviours in two time periods; the initial very short time period where $0 < \tau \ll 1$ during which the enzymes go from being unoccupied to occupied (i.e. $v = 0 \rightarrow v = \frac{u}{u+K}$), and the longer time period, $\tau > 1$ where u gradually decreases to 0.

The trajectory converges to the single stable stationary point at $(0, 0)$, the intersection of the null-clines $du/dt = 0$ and $dv/dt = 0$.

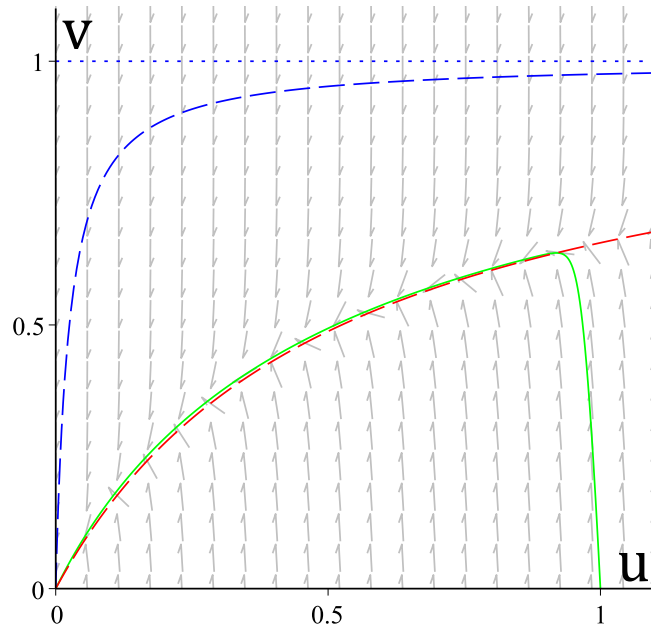


Figure 2.1: Phase Portrait for System 2.20, 2.21

Green line shows a phase trajectory starting at $(1,0)$. The dashed blue line is the null cline $du/dt = 0$. The dashed red line is the null cline $dv/dt = 0$. Horizontal dotted blue line is the asymptote $v = 1$. This particular example was obtained with values $k_{-1} = 1$, $k_1 = 2$, $k_2 = 20$, $s_0 = 20$, $e_0 = 1$. Although not strictly accurate biologically, they serve to obtain a suitable general example of the dynamics.

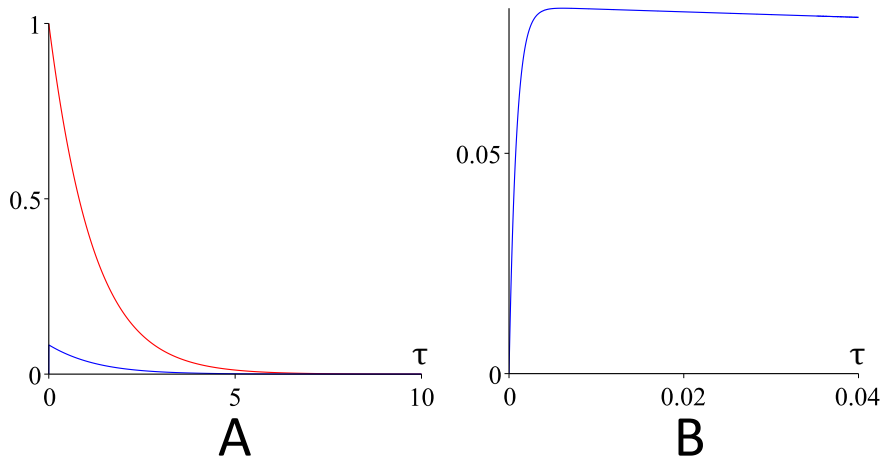


Figure 2.2: Dynamics of variables u and v during enzymatic reaction

A: u (red) and v (blue) plotted against time.

B: detail of **A** for small time values. Notice the rapid increase in the concentration of v over the short initial time period.

In the initial short time period $u \approx 1$ which gives

$$\frac{dv}{d\tau} = \frac{1}{\epsilon}(1 - (1 + K)v) \quad (2.24)$$

$$= \frac{1}{\epsilon} - \frac{1 + K}{\epsilon}v \quad (2.25)$$

Suppose that

$$v(\tau) = A + Be^{-\frac{1+K}{\epsilon}\tau} \quad (2.26)$$

$$\Rightarrow v(\tau) = \frac{1}{1 + K}(1 - e^{-\frac{1+K}{\epsilon}\tau}) \quad (2.27)$$

With $v(0) = 0$, the relaxation time for v is: $\tau_{relv} = \frac{\epsilon}{1+K}$

Then we can find the relaxation time for v using $v = \frac{u}{u+K} = 1 - \frac{K}{u+K}$:

$$\frac{du}{d\tau} = -u + (u + K - \lambda)\left(1 - \frac{K}{u + K}\right) \quad (2.28)$$

$$= -u + u + K - \lambda - \frac{(u + K - \lambda)K}{u + K} \quad (2.29)$$

$$= K - \lambda - K + \frac{\lambda K}{u + K} \quad (2.30)$$

$$= -\frac{\lambda u}{u + K} \quad (2.31)$$

$$\Rightarrow \frac{u + K}{u} du = -\lambda d\tau_r \quad (2.32)$$

$$\Rightarrow u + K \ln u = -\lambda \tau_r + 1 \quad (2.33)$$

Then taking $u(\tau_{relu}) = \frac{1}{e}$:

$$\frac{1}{e} - K = -\lambda \tau_{relu} + 1 \quad (2.34)$$

$$\Rightarrow \tau_{relu} = \frac{1 - 1/e + K}{\lambda} \quad (2.35)$$

We can compare τ_{relu} and τ_{relv} to gain some idea of how long these time periods are in relation to each other.

$$\frac{\tau_{relv}}{\tau_{relu}} = \frac{\epsilon \lambda}{(1 + K)(1 - 1/e + K)} \propto \epsilon \ll 1 \quad (2.36)$$

It is clear from this that the relaxation time for v is extremely small compared to the relaxation time for u .

2.1.3 Quasi-Steady-State Hypothesis

Since the concentration of enzymes is relatively small compared to the concentration of substrate, it is reasonable to assume that after an initial short period since all the enzymes are unoccupied to begin with, no enzyme is ever left unoccupied whilst the reaction is ongoing, i.e. $\frac{dc}{dt} = 0$. This is the Quasi-Steady State Hypothesis.

$$\frac{dv}{d\tau} = 0 \quad (2.37)$$

$$\Rightarrow u - (u + K)v = 0 \quad (2.38)$$

$$\Rightarrow v = \frac{u}{u + K} \quad (2.39)$$

Then, recalling that by definition $c = ve_0$ and $u = \frac{s}{s_0}$, substitute into (2.5):

$$\frac{dp}{dt} = k_2 e_0 \frac{u}{u + K} \quad (2.40)$$

$$\Rightarrow \frac{dp}{dt} = k_2 e_0 \frac{s}{s + K s_0} \quad (2.41)$$

$$\Rightarrow \frac{dp}{dt} = R = \frac{Qs}{s + K_m} \quad (2.42)$$

Where $Q = k_2 e_0$ is the maximal reaction rate and $K_m = \frac{k_{-1} + k_2}{k_1}$, which is known as the Michaelis constant. A general solution for 2.42 is plotted in 2.3.

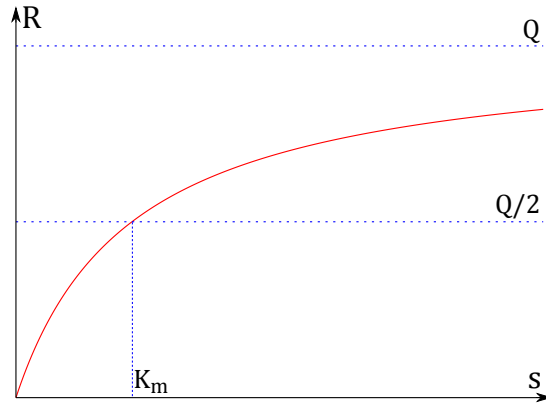


Figure 2.3: Reaction rate as a function of concentration of the substrate

Notice that for $R = \frac{Q}{2}$:

$$\frac{Q}{2} = \frac{k_2 e_0}{2} = k_2 e_0 \frac{s}{s + K_m} \quad (2.43)$$

$$\Rightarrow \frac{1}{2} = \frac{s}{s + K_m} \quad (2.44)$$

$$\Rightarrow s + K_m = 2s \quad (2.45)$$

$$\Rightarrow s = K_m \quad (2.46)$$

2.2 Cooperative Dynamics

A case in which more complex dynamics are observed is where *two* or more molecules of the substrate are required to give a successful reaction.



Which results in similar behaviour with the exception that the graph of the reaction rate with respect to substrate concentration has a sigmoidal shape, for low concentrations, the reaction rate is not linear as it is in the simple case.

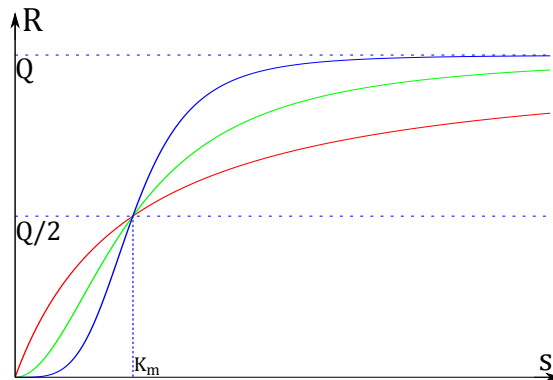


Figure 2.4: Reaction rate as a function of concentration of the substrate

Red - $R(s)$ for the simple case. 2.42

Green - $R(s)$ for cooperative case (requiring two molecules). 2.47

Blue - $R(s)$ for cooperative case (requiring four molecules).

For these cooperative cases, a sigmoidal shape to the graph of reaction rate is obtained, giving what is known as the Hill function. (Murray 1989)[pp.122]

More complex dynamics can be observed if we consider, for example, reactions wherein an intermediary complex C_1 consisting of an enzyme and a substrate molecule can either join with another substrate molecule to give a complex C_2 , or go on to produce a product molecule leaving the original enzyme. The second complex C_2 can then produce a molecule of the C_1 complex with a product.



2.3 Other Types of Dynamical Systems

The case of Michaelis-Menten kinetics is rather simple in that it only considers a system with a single equilibrium with monotonic nullclines, with a single observed behaviour. A greater variety of behaviours can be seen when we consider a non-linear system wherein the dynamics are described by a two-species model:

$$\frac{du}{dt} = f(u, v) \quad (2.50)$$

$$\frac{dv}{dt} = g(u, v) \quad (2.51)$$

Again, this corresponds to a point-system or a solution which is homogenised. If we define $f(u, v) = -\epsilon_u(k_u u(u - u_0)(u - u_1) + v)$ and $g(u, v) = \epsilon_v(u - v)$, where ϵ_u , ϵ_v , k_u , u_0 , and u_1 are constants, we have the FitzHugh-Nagumo Model, a model of an excitable system, most often used to model excitation waves in neurons.

As with any system we can analyse its equilibria by finding the null clines, $f(u, v) = 0$, $g(u, v) = 0$. The equilibria occur at the intersections of the null clines.

For simplicity we only consider $\epsilon_u = 1$ and $\epsilon_v = 0.1$. Depending on the values of k_u , u_0 , and u_1 , we can obtain three general behaviours; oscillatory, excitable or multi-stable.

For an oscillatory system the concentrations of the two chemicals oscillate between two extremes. Examples of this include a stirred Belousov-Zhabotinsky reaction. An example of such a system is shown in figure 2.5.

Excitable behaviour is typical of models used to describe voltage potentials in neurons. Examples include the Hodgkin-Huxley model and the later, simplified version, the FitzHugh-Nagumo model. In figure 2.6, notice that for perturbations less than the threshold value, (here, $u = 0 = 0.15$), the perturbation quickly relaxes to the equilibrium at $(0, 0)$; greater than, they take a much longer time to reset.

In a multi-stable system, the nullclines intersect in more than one place. The example shown in figure 2.7 has two stable equilibria, one at $(0, 0)$ and another at $(\sim 0.693, \sim 0.693)$. A third, unstable equilibrium can be found at $(\sim 0.45, \sim 0.45)$ at the third intersection, though since this one is unstable, it repels any nearby orbits so hence is generally not observable. Multi-stable systems can be realised as biochemical switches, wherein a cell switches between two stable states or behaviours.

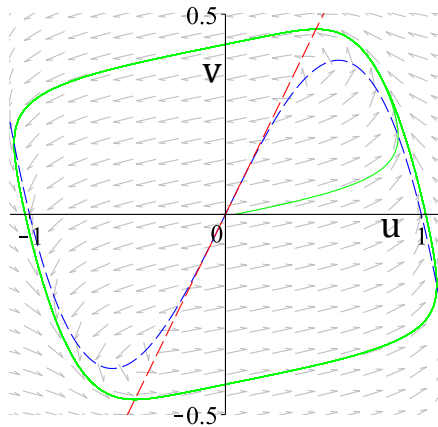


Figure 2.5: Oscillatory behaviour in the FitzHugh-Nagumo system

Here for demonstrative purposes $u_0 = -1$, $u_1 = 1$, $k_u = 4.5$. The blue dashed line shows the null cline $du/dt = 0$, the red dashed line the null cline $dv/dt = 0$. The green line is an example trajectory starting at $(0.05, 0)$, a small perturbation from the single unstable equilibrium at $(0, 0)$. The orbit which it falls onto is the limit cycle for this system.

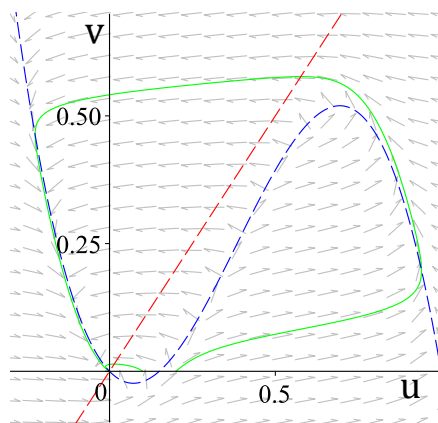


Figure 2.6: Excitable behaviour in the FitzHugh-Nagumo system

Here we have $u_0 = 0.15$, $u_1 = 1$, $k_u = 5$. There is only one equilibrium which is stable. Again, the blue dashed line shows the null cline $du/dt = 0$, the red dashed line the null cline $dv/dt = 0$. The green lines here show two trajectories; one starting at $(0.1, 0)$ which quickly returns to the equilibrium at $(0, 0)$, while the other starting at $(0.2, 0)$ makes an extended tour before returning to the equilibrium.

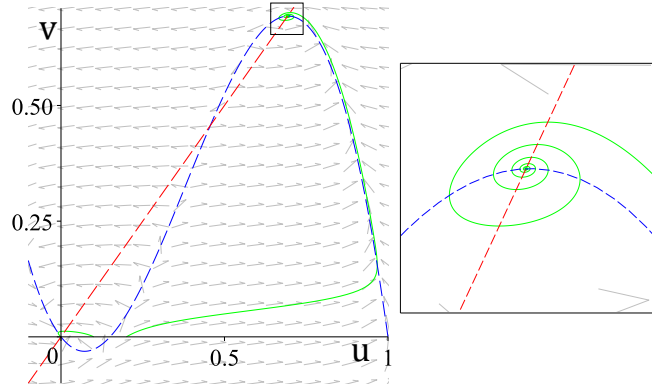


Figure 2.7: Multistable behaviour in the FitzHugh-Nagumo system

Here we have $u_0 = 0.15$, $u_1 = 1$, $k_u = 6$. There are now three equilibria of which two are stable, one is unstable. Again, the blue dashed line shows the null cline $du/dt = 0$, the red dashed line the null cline $dv/dt = 0$. The green lines again show two trajectories; starting at $(0.1, 0)$ which quickly returns to the equilibrium at $(0, 0)$, while the other starting at $(0.2, 0)$ is attracted to the other stable equilibrium at $(\sim 0.693, \sim 0.693)$. Detail of the spiraling trajectory is shown in the right.

2.4 Distributed Systems

Furthermore, we can also consider these reactions taking place within a space with the addition of diffusion to distribute it.

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + f(u, v) \quad (2.52)$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + g(u, v) \quad (2.53)$$

These systems where both chemical reactions and diffusion are contributing to the dynamics are known as reaction-diffusion systems. In such systems it is common to observe patterns. For instance, if we apply a diffusion terms to the FitzHugh-Nagumo model above we can observe propagating waves, spiral waves and stationary spots. This will be discussed in greater detail in the chapter 4.

When these patterns are observed, with a change in size of the medium in which the reaction is taking place, the patterns can scale with the size of the medium. However, this scaling is often not proportional to the change in size as we might expect, instead the patterns are distorted in some way. This problem of scaling will be discussed in the next chapter.

Chapter 3

Diffusion-Decay Systems

In a biological system patterns are often scaled with the size of the organism. For instance, a small zebra that grows into a larger zebra does not have stripes the same width, they are proportional to its size. However models such as those proposed by Turing and Meinhardt, the size of stripes in a pattern are dictated by the Diffusion coefficient and rate of reaction, and not affected by the size of the system (Ishihara and Kaneko 2006).

In the previous section we discussed a point system. Although our main concern was with enzymatic reactions, this can be generalised to some generic morphogen.

It is reasonable to think of such a chemical being produced, diffusing through a tissue and decaying. In this section we will investigate the effect of the size of a medium on the chemical gradient of a single morphogen diffusing through a length of tissue. We do this by calculating the relative position, ξ , within a medium, that is, the position relative to the size of the medium, L , such that $\xi = 0$ is the left hand boundary, $\xi = 1$ the right, and comparing the concentration of the morphogen across the rescaled media. Furthermore, we will derive the so-called scaling factor, S , which serves as a measure of the distortion of the gradient with a change in length of the medium.

3.1 Solutions for Diffusion-Decay Systems

We consider the following equation of a morphogen u diffusing with coefficient D and decaying at a rate k :

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} - ku \quad (3.1)$$

For simplicity, we consider only the stationary solution, i.e. with $\partial u / \partial t = 0$, so that the equation becomes:

$$D \frac{d^2 u}{dx^2} - ku = 0 \quad (3.2)$$

for $x \in [0, L]$ with L being the length of the medium. The general solution of this is:

$$u = C_1 e^{\sqrt{\frac{k}{D}}x} + C_2 e^{-\sqrt{\frac{k}{D}}x} \quad (3.3)$$

With constants C_1 and C_2 defined by the boundary conditions.

For convenience, define $\lambda = \sqrt{D/k}$:

$$u = C_1 e^{\frac{x}{\lambda}} + C_2 e^{-\frac{x}{\lambda}} \quad (3.4)$$

The boundary conditions are defined either as Dirichlet, Neumann or a combination of the two:

Dirichlet Boundary Conditions - defining fixed concentrations at the boundaries. Biologically this would represent a source of the chemical u to the left of the medium and a sink to the right i.e. something producing the chemical u in such a way that should the concentration drop below a certain level, it produces more. Should it rise above this level it ceases production so that the concentration is constant. At right it is simply being consumed as fast as possible so that the concentration is always 0.

Neumann Boundary Conditions - defining the flux at the boundaries. Rather than the morphogen being produced in such a way that its concentration remains constant, the *rate* of production is constant. To the right of the medium, we define the flux as zero, i.e., there is no sink.

Mixed Boundary Conditions - Fixed concentration at left hand boundary $\xi = 0$ and fixed flux at right hand boundary $\xi = 1$, that is, a dynamic source as in the Dirichlet case to the left and no sink at right.

3.1.1 Dirichlet Boundary Conditions

Taking $u(0) = 1$ and $u(L) = 0$ we obtain:

$$u = \frac{e^{\frac{x}{\lambda}} - e^{\frac{2L-x}{\lambda}}}{1 - e^{\frac{2L}{\lambda}}} \quad (3.5)$$

Then, defining the *relative position* $\xi = \frac{x}{L}$:

$$u = \frac{e^{\frac{\xi L}{\lambda}} - e^{\frac{2-\xi}{\lambda}L}}{1 - e^{\frac{2L}{\lambda}}} \quad (3.6)$$

Sample solutions for the profile of u are shown in 3.1 **A**.

3.1.2 Neumann Boundary Conditions

Taking $u'(0) = -q$ and $u'(L) = 0$, q being the boundary flux at $x = 0$, we obtain:

$$u = q\lambda \frac{e^{\frac{x}{\lambda}} + e^{\frac{2L-x}{\lambda}}}{1 - e^{\frac{2L}{\lambda}}} \quad (3.7)$$

Again, substituting *relative position* ξ :

$$u = q\lambda \frac{e^{\frac{\xi L}{\lambda}} + e^{\frac{2-\xi}{\lambda}L}}{1 - e^{\frac{2L}{\lambda}}} \quad (3.8)$$

Some solutions are illustrated in 3.1 **B**. For simplicity we take $q = 1$.

3.1.3 Mixed Boundary Conditions

Taking $u(0) = 1$ and $u'(L) = 0$, we obtain:

$$u = \frac{e^{\frac{x}{\lambda}} + e^{\frac{2L-x}{\lambda}}}{e^{\frac{2L}{\lambda}} + 1} \quad (3.9)$$

Again, substituting *relative position* ξ :

$$u = \frac{e^{\frac{\xi L}{\lambda}} + e^{\frac{2-\xi}{\lambda}L}}{e^{\frac{2L}{\lambda}} + 1} \quad (3.10)$$

Again, some solutions for u are illustrated in 3.1 **C**.

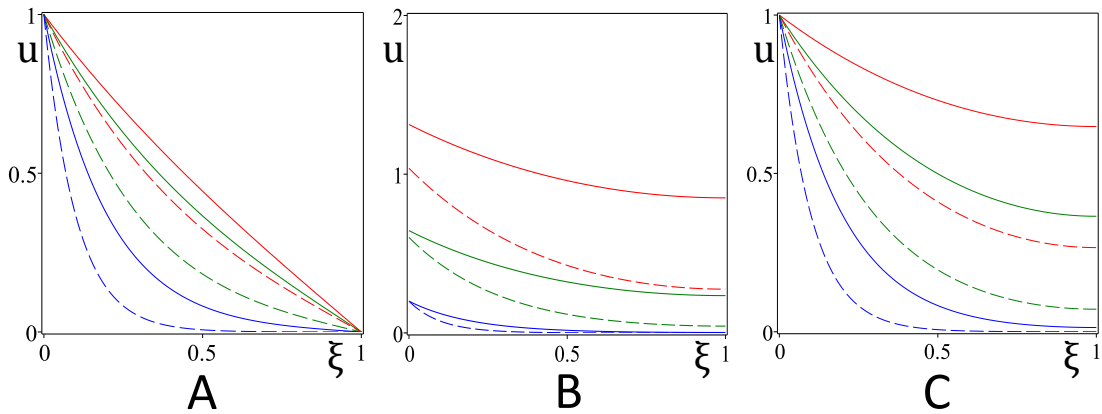


Figure 3.1: Profiles of u for different boundary conditions

In each of **A**, **B**, **C**: $L = 1, 2$ - solid, dashed lines respectively. $\lambda = 1, 0.6, 0.2$ - red, green, blue lines respectively.

A: profile of u for Dirichlet BCs.

B: profile of u for Neumann BCs.

C: profile of u for mixed BCs.

When we compare the profiles of u for these scaled media, it is clear that profile is not scaled with the medium.

In the case of the Dirichlet boundary conditions (Fig 3.1 **A**), the simple case where $L = 1$ and $\lambda = 1$, we obtain a linear profile. At either end of the medium, since this is the manner in which we defined the boundary conditions, all the profiles coincide. However across the rest of the medium with increasing L and decreasing λ , the profiles deviate further and further from the linear profile, becoming increasingly exponential.

With Neumann boundary conditions, the opposite behaviour is observed; with decreasing values of λ , the profiles deviate less with a change in L .

For the Mixed boundary conditions, at the left hand boundary, the profiles all coincide as expected, and again deviate with increasing L and decreasing λ .

Although we can quite clearly observe the scaling (or lack thereof) from Fig. 3.1, it would be more instructive to measure how much the profiles are distorted in each case.

3.2 Scaling in Diffusion-Decay Systems

3.2.1 Defining Scaling Factor

In order to quantify the distortion of the chemical gradient it is necessary to define some quantity to describe it. We use the definition described in (Rasolonjanahary 2013), the derivation of which is described below.

Suppose we have two different-sized mediums with lengths L_1 and L_2 . If there is “good” scaling (i.e. no distortion) of the morphogen gradient between the two media, $u(\xi, L) = u(\xi, L + \Delta L)$ for any ξ along the length. As we can see above in Fig 3.1, in general this is not the case.

Suppose then that $u(\hat{\xi}, L) = u(\hat{\xi} + \Delta\xi, L + \Delta L)$ (See Fig. 3.2).

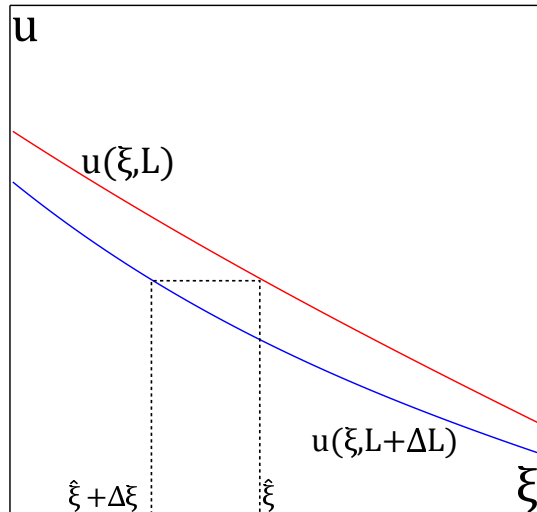


Figure 3.2: Derivation of the Scaling Factor

Two non-identical profiles of u for differing lengths of of the medium, L and $L + \Delta L$. The concentrations are equal at $u(\hat{\xi}, L) = u(\hat{\xi} + \Delta\xi, L + \Delta L)$

Then for small changes in ξ and L we have the first order approximation

$$u(\hat{\xi}, L) = u(\hat{\xi} + \Delta\xi, L + \Delta L) + \frac{\partial u}{\partial \xi}((\hat{\xi} + \Delta\xi) - \hat{\xi}) + \frac{\partial u}{\partial L}((L + \Delta L) - L) \quad (3.11)$$

$$\frac{\partial u}{\partial \xi} \Delta\xi + \frac{\partial u}{\partial L} \Delta L = 0 \quad (3.12)$$

$$\Rightarrow \Delta\xi = -\frac{\frac{\partial u}{\partial L}}{\frac{\partial u}{\partial \xi}} \Delta L \quad (3.13)$$

So for a change in L , ΔL , the associated change in ξ , $\Delta\xi$ is proportional to $-\frac{\partial u}{\partial L} \left(\frac{\partial u}{\partial \xi}\right)^{-1}$
We define

$$S(\xi) = -\frac{\partial u}{\partial L} \left(\frac{\partial u}{\partial \xi}\right)^{-1} \quad (3.14)$$

as the 'Scaling Factor', a measure of how the chemical gradient is distorted with a change in the size of the medium, L .

When the scaling factor is negative, the relative position with the same value for u is to the left, when it is positive, it is to the right. When it is zero, the scaling is perfect.

Applying this formula for S to the three formulae for u we obtain:

For Dirichlet Boundary Conditions:

$$S = -\frac{(1 - e^{\frac{2L}{\lambda}})(\xi e^{\frac{\xi L}{\lambda}} - (2 - \xi)e^{\frac{2-\xi}{\lambda}L}) + 2e^{\frac{2L}{\lambda}}(e^{\frac{\xi L}{\lambda}} - e^{\frac{2-\xi}{\lambda}L})}{L(1 - e^{\frac{2L}{\lambda}})(e^{\frac{\xi L}{\lambda}} - e^{\frac{2-\xi}{\lambda}L})} \quad (3.15)$$

For Neumann Boundary Conditions:

$$S = -\frac{(e^{\frac{2L}{\lambda}} - 1)(\xi e^{\frac{\xi L}{\lambda}} + (2 - \xi)e^{\frac{2-\xi}{\lambda}L}) - 2e^{\frac{2L}{\lambda}}(e^{\frac{\xi L}{\lambda}} + e^{\frac{2-\xi}{\lambda}L})}{L(e^{\frac{2L}{\lambda}} - 1)(e^{\frac{\xi L}{\lambda}} - e^{\frac{2-\xi}{\lambda}L})} \quad (3.16)$$

For Mixed Boundary Conditions:

$$S = -\frac{(e^{\frac{2L}{\lambda}} + 1)(\xi e^{\frac{\xi L}{\lambda}} + (2 - \xi)e^{\frac{2-\xi}{\lambda}L}) - 2e^{\frac{2L}{\lambda}}(e^{\frac{\xi L}{\lambda}} + e^{\frac{2-\xi}{\lambda}L})}{L(e^{\frac{2L}{\lambda}} + 1)(e^{\frac{\xi L}{\lambda}} - e^{\frac{2-\xi}{\lambda}L})} \quad (3.17)$$

These three formulae for S were calculated by hand then checked using Maple, which gave the same results, albeit with some slight differences due to factorisation.

Some solutions of S for the three different boundary conditions are plotted in 3.3. Note that since S is generally negative, for simplicity we plot the absolute value $|S|$.

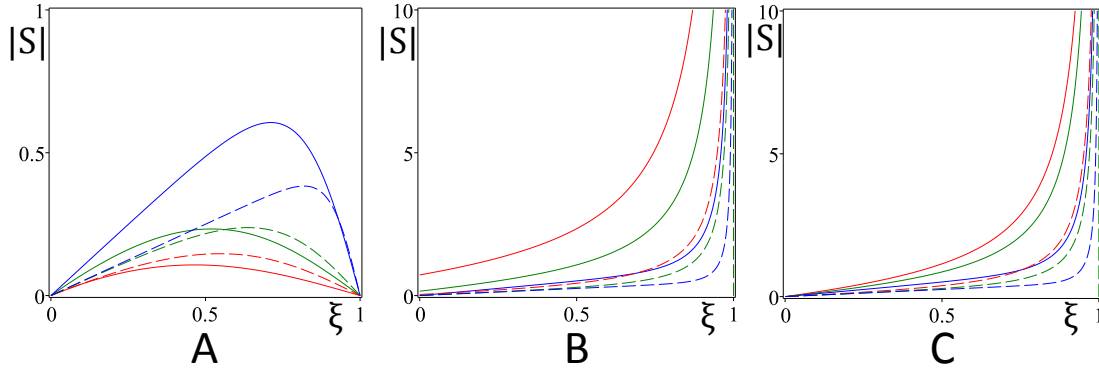


Figure 3.3: Scaling factor, $|S|$ as a function of relative position for three sets of boundary conditions

A: $|S(\xi)|$ for Dirichlet BCs.

B: $|S(\xi)|$ for Neumann BCs.

C: $|S(\xi)|$ for mixed BCs.

In each of **A**, **B**, **C**:

$L = 1, 2$ - solid, dashed lines respectively.

$\lambda = 1, 0.6, 0.2$ - red, green, blue lines respectively.

Comparing the profiles of u in fig 3.1 with the respective plots for scaling factor in fig 3.3 we observe that where the values for u coincide, i.e. the scaling is perfect, the scaling factor $S = 0$. For negative values of the scaling factor the value of ξ in the rescaled medium with the same value of u is to the left. For positive values, it is to the right.

In the case of Neumann and Mixed boundary conditions the scaling improves (i.e. the scaling factor decreases) with an increased medium size along the entire medium, shown in Fig 3.3 **B** and **C**, which corresponds with what we observed earlier.

However in the case of Dirichlet boundary conditions, with an increase in medium size, at some points along the medium the scaling increases, at others it decreases. This can be seen in Fig 3.3 **A** where for each value of λ , the dotted and solid lines intersect at some point.

3.2.2 Comparing Scaling Factors for Neumann and Dirichlet Boundary Conditions

If we take L as a constant value (say, $L = 1$), then it is possible to observe the limiting behaviour of the scaling factor for Dirichlet and Neumann boundary conditions with a decreasing λ .

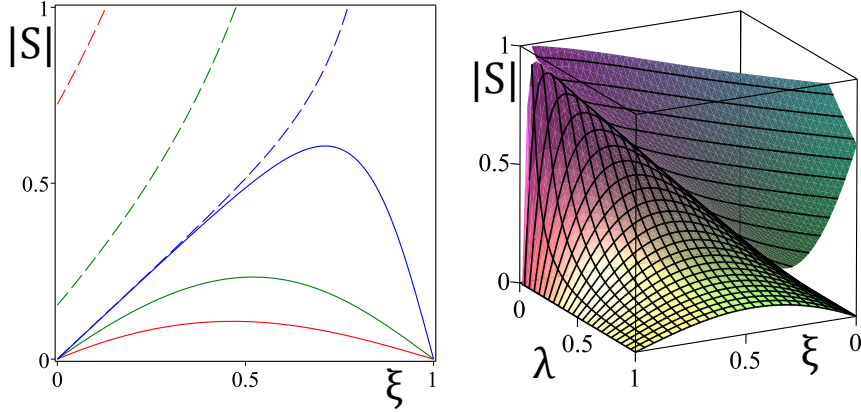


Figure 3.4: Scaling Factor for Dirichlet and Neumann boundary conditions for $L = 1$

Left: Dirichlet Boundary Conditions - solid line

Neumann Boundary Conditions - dashed line

$\lambda = 1, 0.6, 0.2$ (red, green and blue lines respectively)

This also serves as a cross-section of the plot at right.

Right: 3-dimensional plot, Dirichlet Boundary conditions give the lower surface, Neumann the upper. With decreasing λ the two surfaces tend toward the line $S = \xi$. It was necessary to reverse the direction of the ξ -axis (i.e. with $\xi = 1$ on the left, $\xi = 0$ on the right) in order to properly observe this behaviour in a non-interactive way.

Notice that although both plots converge to the line $S = \xi$, they never intersect. This implies that:

$$|S_{Neumann}| > |S_{Dirichlet}| \quad \forall \lambda, L, \xi \quad (3.18)$$

3.2.3 Effect of Medium Size on Scaling Factor under Dirichlet Boundary conditions

It is interesting to see the transition point at which the scaling goes from being improved to reduced. This transition point is the point at which, for example the plots for $S(L = 1)$ and $S(L = 2)$ intersect, that is, $S(\xi, L = 1) = S(\xi, L = 2)$. We can study the behaviour of this intersection qualitatively by finding the point for any given λ where $\frac{\partial S}{\partial L} = 0$.

Then, calculating $\frac{\partial S}{\partial L}$ and plotting (implicitly) $\frac{\partial S}{\partial L} = 0$ for constant L we obtain Fig. 3.5 A.

$$\exists \chi \in [0, 1] : \left. \frac{\partial S}{\partial L} \right|_{\xi=\chi} = 0$$

Where the value of χ is dependent on L and λ .

It is obvious that

$$\lim_{\lambda \rightarrow 0} \chi = 1 \quad (3.19a)$$

$$\lim_{\lambda \rightarrow \sim 0.75} \chi = 0 \quad (3.19b)$$

We can also do the same for a constant λ and changing L to obtain the following:

$$\lim_{L \rightarrow 1} \chi = 0 \quad (3.20a)$$

$$\lim_{L \rightarrow \infty} \chi = 1 \quad (3.20b)$$

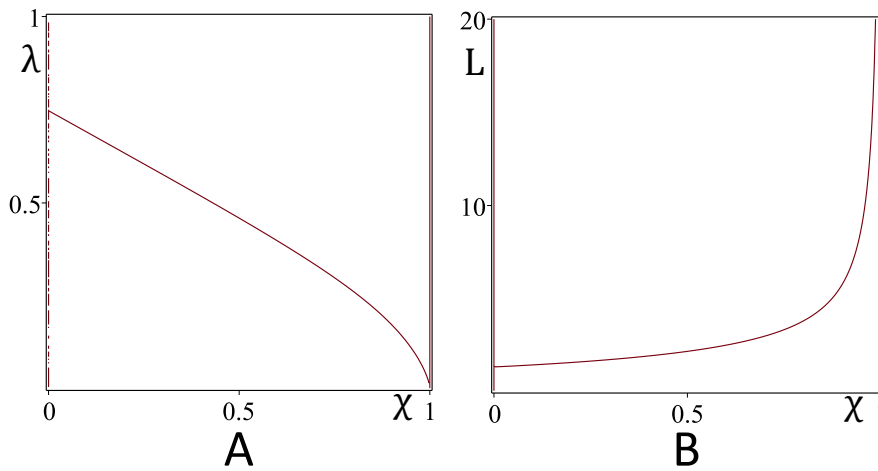


Figure 3.5: Behaviour of χ , the point at which scaling is not changing for Dirichlet boundary conditions.

A: χ for constant L (here, $L = 1$) with $\lambda \in (0, 1)$.

B: χ for constant λ (here, $\lambda = 1$) with $L \in (0, 20)$.

The formulae for $\frac{\partial S}{\partial L}$ were calculated using Maple, then $S_L = 0$ plotted implicitly.

Then we can say that, under Dirichlet boundary conditions, for any given λ and the corresponding value of χ :

For $\xi < \chi$, with increasing L , scaling improves.

For $\xi > \chi$, with increasing L , scaling worsens.

Chapter 4

Non-linear Reaction-Diffusion Systems

4.1 Models

Two-Component Reaction-Diffusion systems are generally of the form

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + f(u, v) \quad (4.1)$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + g(u, v) \quad (4.2)$$

In the case where the chemical u promotes its own production as well as the production of v , and v inhibits the production of u , u is referred to as the activator and v as the inhibitor.

With differing definitions of f and g we obtain different models. For instance with

$$f(u, v) = \rho \frac{u^2}{v} - \mu_u u + \rho_u \quad (4.3)$$

$$g(u, v) = \rho u^2 - \mu_v v + \rho_v \quad (4.4)$$

with constants $\rho, \mu_u, \mu_v, \rho_u, \rho_v$ we have the model proposed by Gierer and Meinhardt, a general model which can be used to model the formation of periodic patterns.

With f, g defined as:

$$f(u, v) = -\epsilon_u (k_u u (u - u_o)(u - u_1) + v) \quad (4.5)$$

$$g(u, v) = \epsilon_v (u - v) \quad (4.6)$$

We have an altered version of the Fitzhugh-Nagumo model (Vasiev 2004). The Fitzhugh-Nagumo model was originally developed to model excitation waves in neurons, but has been adapted with the addition of diffusion terms to become a reaction-diffusion

system.

4.2 Fitzhugh-Nagumo

The simulations here are created with a initial conditions of $u = 0.1, v = 0.1$ and a small disturbed area around the centre of the medium ($u = 0.5$). Unless otherwise noted, in the following examples, $D_u = 1, k_u = 4.5, u_0 = 0.05, u_1 = 1, \epsilon_u = 1, \epsilon_v = 1$.

Videos of all simulations shown in this chapter can be found online at <http://pcwww.liv.ac.uk/~sgaphill/>.

4.2.1 Travelling Wave

For $D_v = 1$ we have a travelling wave which starts at the centre of the medium and propagates outwards toward the boundaries.

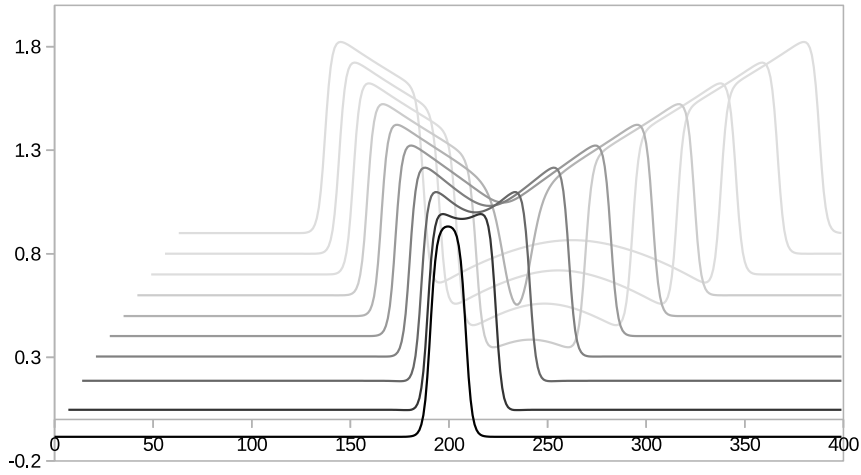


Figure 4.1: Travelling Wave in the FHN model

Only the concentration of the activator is shown here for clarity. The initial spot splits into a pair of propagating waves. In this figure, profiles at later points in time are shown in decreasingly paler shades of grey, recessed in a pseudo-z-axis.

Waves continue to be observed when D_v is sufficiently small, e.g. when $\epsilon_v = 0.1$, propagating waves are observed for $D_v < 1.8$ (Vasiev 2004).

4.2.2 Pulsating Spot

At $D_v = 3$ the system oscillates stably, illustrated in fig 4.2. The position of the spot does not change, but it oscillates in width as well as in amplitude.

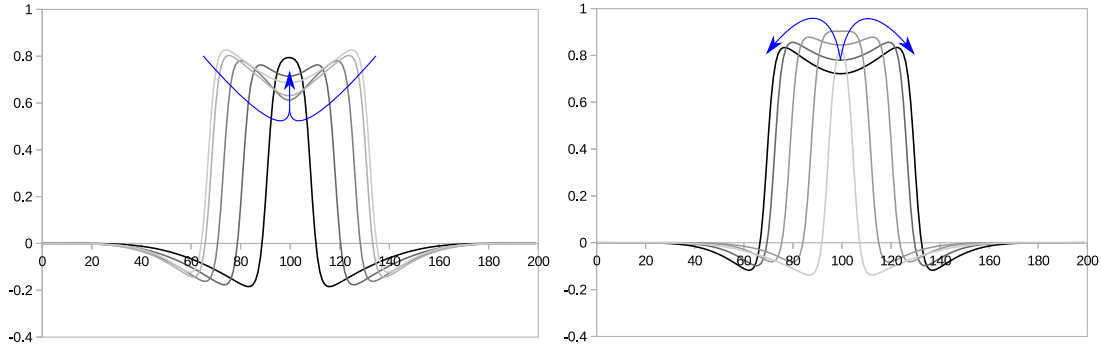


Figure 4.2: Pulsating Spot in the FHN model

Only the concentration of the activator is shown here for clarity. Notice that the spot oscillates in amplitude as well as width. The profiles for later points in time are in darker shades of grey.

Pulsating spots are observed when D_v lies within some range which is dictated by ϵ_v . For instance, for $\epsilon_v = 0.1$, pulsating spots are observed when $3 < D_v < 3.5$ (Vasiev 2004).

4.2.3 Stationary Spot

Then for $D_v = 4$, the system oscillates to a stable non-homogeneous equilibrium, a stationary spot. Again the position of the spot does not change, but the oscillations dampen down to this stable equilibrium. Larger values of D_v give wider spots.

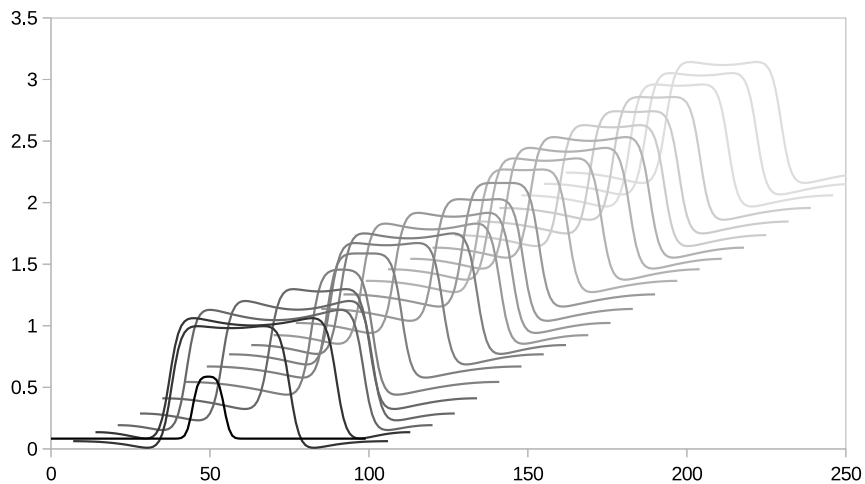


Figure 4.3: Stationary spots in the FitzHugh-Nagumo System

$D_v = 4$ - The spot oscillates to a stable equilibrium. Profiles for greater values of time are shown in paler shades of grey.

With changing ϵ_v , provided D_v is sufficiently large, stationary spots continue to be observed. For example, for $\epsilon_v = 0.1$, stationary spots are observed when $D_v > 3.5$ (Vasiev 2004).

4.3 Stability Analysis of the Stationary Spot

In order to analyse the stability of this spot, it is sufficient to consider the profile of the activator as an approximation of the form of $u = 1$ for $|x| < a$ and $u = 0$ for $|x| > a$ where $2a$ is the width of the spot. We then find the corresponding approximation of v .

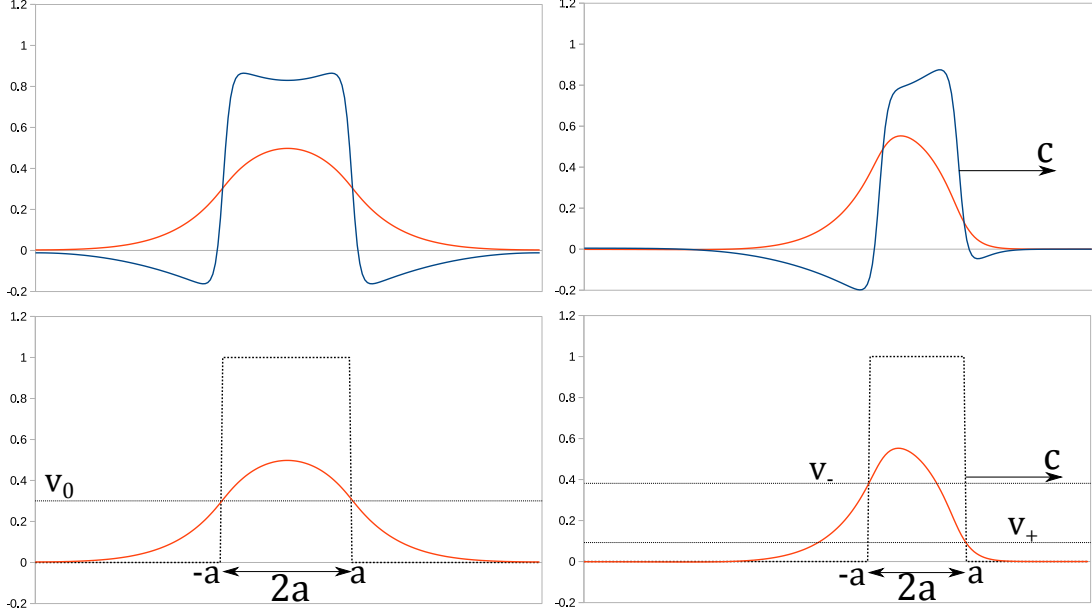


Figure 4.4: Approximation of the stationary spot and travelling wave
Replacing the profile for u with the rectangular function $u = 1$ for $|x| \leq a$, and $u = 0$ for $|x| > a$ where $2a$ is the width of the spot.

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} - \epsilon_v(u - v) \quad (4.7)$$

Since we're considering the stationary spot, we can say that $\frac{\partial v}{\partial t} = 0$ giving:

$$D_v \frac{d^2 v}{dx^2} - \epsilon_v(u - v) = 0 \quad (4.8)$$

Our boundary conditions are as follows:

$$v(\infty) = v(-\infty) = 0 \quad (4.9)$$

$$v(a) = v(-a) = v_0 \quad (4.10)$$

With some constant concentration v_0 . Solving this ODE we obtain:

$$v(x) = \begin{cases} 1 + (v_0 - 1) \frac{\cosh(\lambda x)}{\cosh(\lambda a)} & : |x| \leq a \\ v_0 \exp(\lambda(a - |x|)) & : |x| > a \end{cases} \quad (4.11)$$

where $\lambda = \sqrt{\epsilon_v/D_v}$.

If we require that $v'(x)$ is continuous at $x = \pm a$ then we obtain the relationship $\lambda a = \frac{1}{2} \ln(1 - 2v_0)^{-1}$.

Furthermore the maximum value of v at the centre of the spot is:

$$v(0) = 1 + \sqrt{1 - 2v_0} \quad (4.12)$$

In order to analyse the stability, we consider a perturbation of the steady state (a_0, v_0) .

$$\frac{da}{dt} = f(a, v) \quad (4.13)$$

$$\frac{dv}{dt} = g(a, v) \quad (4.14)$$

Where f and g are functions to be determined.

Then introducing a perturbation:

$$\frac{d}{dt}(a_0 + \delta a) = \frac{d\delta a}{dt} = f(a_0 + \delta a, v_0 + \delta v) \quad (4.15)$$

$$\frac{d}{dt}(v_0 + \delta v) = \frac{d\delta v}{dt} = g(a_0 + \delta a, v_0 + \delta v) \quad (4.16)$$

Which gives the first-order linearisation:

$$\frac{d\delta a}{dt} = f(a_0, v_0) + f_a(a_0, v_0)\delta a + f_v(a_0, v_0)\delta v \quad (4.17)$$

$$\frac{d\delta v}{dt} = g(a_0, v_0) + g_a(a_0, v_0)\delta a + g_v(a_0, v_0)\delta v \quad (4.18)$$

And since (a_0, v_0) is the steady state, $f(a_0, v_0) = g(a_0, v_0) = 0$, so:

$$\frac{d\delta a}{dt} = f_a(a_0, v_0)\delta a + f_v(a_0, v_0)\delta v \quad (4.19)$$

$$= b_{11}\delta a + b_{12}\delta v \quad (4.20)$$

$$\frac{d\delta v}{dt} = g_a(a_0, v_0)\delta a + g_v(a_0, v_0)\delta v \quad (4.21)$$

$$= b_{21}\delta a + b_{22}\delta v \quad (4.22)$$

$$\text{With } B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} f_a & f_v \\ g_a & g_v \end{pmatrix}$$

$$\begin{pmatrix} \delta a \\ \delta v \end{pmatrix}' = B \begin{pmatrix} \delta a \\ \delta v \end{pmatrix} \quad (4.23)$$

B is the Jacobian of the system; we can analyse stability by looking at the trace, $tr(B)$, and the determinant, $det(B)$. The point (a_0, v_0) is a stable for $tr(B) < 0$ and $det(B) > 0$, and unstable for $tr(B) > 0$ and $det(B) > 0$.

The transition between stable and unstable states occurs when

$$tr(B) = b_{11} + b_{22} = 0 \quad (4.24)$$

so it is of most interest to us to find the conditions at this transition.

da/dt represents a change in the boundary of the spot (or wave), which we can take as the speed $c(v)$ of the boundary.

$c(v)$ is proportional to the integral of the function $f(u, v)$ between the maximal and minimal roots of the equation $f(u, v) = 0$ (Vasiev 2004).

$$c(v) \propto \int_{u_-}^{u_+} f(u, v) du \quad (4.25)$$

Where u_+ and u_- are the maximal and minimal roots respectively, shown for $f(u, v_0) = 0$ in fig 4.5.

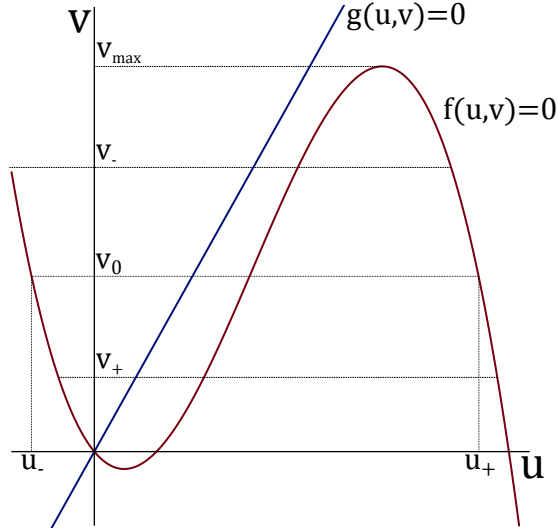


Figure 4.5: Nullclines $f(u, v) = 0$, $g(u, v) = 0$ for the Fitzhugh-Nagumo system

So as v increases, c decreases, and vice versa, i.e. $c'(v) < 0$. Since v is a function of a

we have:

$$\frac{da}{dt} = c(v(a)) \quad (4.26)$$

Then at the boundary of the stationary spot where $v = v_0$, $c(v_0) = 0$ (since the spot is stationary), we have the Taylor expansion:

$$c(v_0 + \delta v) \approx c(v_0) + c'(v_0)\delta v \quad (4.27)$$

$$= c'(v_0)\delta v \quad (4.28)$$

Then for δv

$$\delta v = v(a + \delta a) - v(a) \quad (4.29)$$

Taking a Taylor expansion on $v(a + \delta a)$, we have:

$$\delta v \approx [v(a) + v'(a)\delta a] - v(a) = v'(a)\delta a \quad (4.30)$$

So we have:

$$c(v_0 + \delta v) \approx c'(v_0)v'(a)\delta a \quad (4.31)$$

$$\Rightarrow b_{11} = c'(v_0)v'(a) \quad (4.32)$$

And then calculating $v'(a) = -\lambda v_0$ from 4.11:

$$b_{11} = -\lambda v_0 c'(v_0) \quad (4.33)$$

Then in order to find b_{22} we consider dv/dt :

$$\frac{d}{dt}(v_0 + \delta v) = D_v \Delta(v_0 + \delta v) + \epsilon_v(u_0 - [v_0 + \delta v]) \quad (4.34)$$

$$= D_v \Delta v_0 + \epsilon_v(u_0 - v_0) + D_v \Delta \delta v - \epsilon_v \delta v \quad (4.35)$$

$$= \underbrace{\frac{dv_0}{dt}}_{=0} + \frac{d\delta v}{dt} \quad (4.36)$$

Where Δ is the Laplacian operator.

The spatial distribution can be represented as a fourier series,

$$\delta v = \sum_{k=0}^{\infty} e^{\gamma(k)t} \cos kx \quad (4.37)$$

$$\Rightarrow \frac{d\delta v}{dt} = \frac{d}{dt} e^{\gamma(k)t} \cos(kx) = D_v \Delta e^{\gamma(k)t} \cos kx - \epsilon_v e^{\gamma(k)t} \cos kx \quad (4.38)$$

$$= -D_v k^2 e^{\gamma(k)t} \cos kx - \epsilon_v e^{\gamma(k)t} \cos kx \quad (4.39)$$

$$= -(D_v k^2 + \epsilon_v) e^{\gamma(k)t} \cos kx \quad (4.40)$$

$$= -(D_v k^2 + \epsilon_v) \delta v \quad (4.41)$$

Which means that $b_{22} = -(D_v k^2 + \epsilon_v)$ so that:

$$tr(B) = b_{11} + b_{22} \quad (4.42)$$

$$= -\lambda v_0 c'(v_0) - (D_v k^2 + \epsilon_v) \quad (4.43)$$

Which is at its largest, and therefore most sensitive where $k = 0$. So the condition $tr(B) > 0$ becomes

$$tr(B) = -\lambda v_0 c'(v_0) - \epsilon_v < 0 \quad (4.44)$$

And recalling $\lambda = \sqrt{\epsilon_v/D_v}$ we have the relation

$$\Rightarrow D_v < \frac{v_0^2}{\epsilon_v} c'(v_0)^2 \quad (4.45)$$

for stability.

Then for the condition on the determinant, we must find expressions for b_{12} and b_{21} .

For b_{12} :

$$\frac{da}{dt} = c(v) \approx c(v_0 + \delta v) \quad (4.46)$$

$$= \underbrace{c(v_0)}_{=0} + c'(v_0) \delta v \quad (4.47)$$

so $b_{12} = c'(v_0)$

Then for b_{21} :

$$b_{21}\delta a = \epsilon_v(v_0(a + \delta a) - v(a + \delta a)) \quad (4.48)$$

$$= \epsilon_v((v_0 + (1 + 2v_0)\lambda\delta a) - v_0(1 - \lambda\delta a)) \quad (4.49)$$

$$= \epsilon_v(\lambda\delta a - 2v_0\lambda\delta a + v_0\lambda\delta a) \quad (4.50)$$

$$= \epsilon_v(\lambda\delta a - v_0\lambda\delta a) \quad (4.51)$$

$$= \epsilon_v\lambda(1 - v_0)\delta a \quad (4.52)$$

$$\Rightarrow b_{21} = \epsilon_v\lambda(1 - v_0) \quad (4.53)$$

So the condition $\det(B) > 0$ becomes:

$$\lambda v_0 c'(v_0)(D_v k^2 + \epsilon_v) - c'(v_0)\epsilon_v(1 - v_0)\lambda > 0 \quad (4.54)$$

$$(4.55)$$

Then with mode $k = 0$ again:

$$\Rightarrow \epsilon_v\lambda v_0 c'(v_0) - c'(v_0)\epsilon_v\lambda + c'(v_0)\epsilon_v v_0\lambda > 0 \quad (4.56)$$

And with cancellation of λ , ϵ_v , $c'(v_0)$, remembering that $c'(v_0) < 0$ we have the condition:

$$v_0 < 0.5 \quad (4.57)$$

Crucially, the condition of $\det(B) > 0$ depends only v_0 , not on D_v or ϵ_v .

So our conditions for stability are:

$$D_v < \frac{v_0^2}{\epsilon_v} c'(v_0)^2 \propto \frac{1}{\epsilon_v} \quad (4.58)$$

$$v_0 < 0.5 \quad (4.59)$$

4.4 Fitzhugh-Nagumo with noise

Here initial conditions were $u = 0$, $v = 0$ across the whole medium. Noise was applied every 500 timesteps for 100 timesteps in small random areas with small random values (in the range of $(-0.02, 0.02)$).

Depending on the value used for D_v , as in the simple case above, we can observe different behaviour.

Again, for value of $D_v < 3$ travelling waves are observed. However due to the nature of the noise, multiple waves will often tend to occur at once within the medium. The waves will be moving in opposite directions will interact with each other.

For $D_v > 3$, stationary spots are again observed for perturbations of sufficient magnitude. Otherwise the perturbations will simply die down to the homogeneous state.

Of most interest is the situation in which some perturbations cause excitations and others do not.

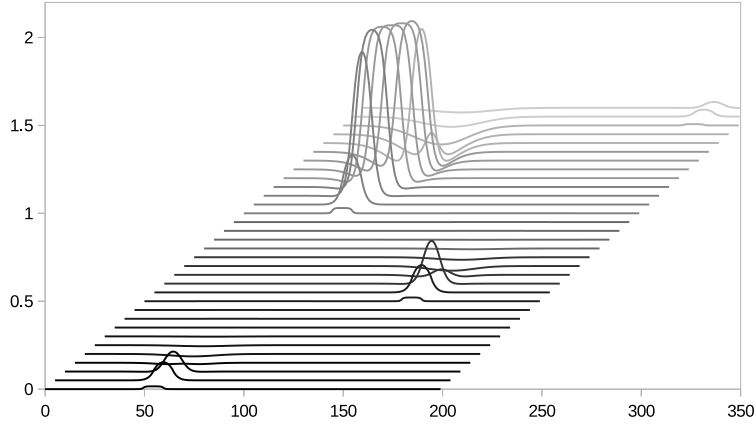


Figure 4.6: Noise in the FitzHugh-Nagumo Model

An example of noise in the FitzHugh-Nagumo Model. Small random perturbations. $D_v = 2$, $u_0 = 0.15$, $\epsilon_v = 0.2$. The smaller perturbations quickly returns back to the homogenous state, whilst the larger ones excite for a moment. Compare this to the behaviour observed in the first chapter for an excitable FitzHugh-Nagumo system (see Fig. 2.6) where a sufficiently large perturbation from the stable equilibrium will result in an excitation.

Profiles for greater values of time are shown in paler shades of grey.

A possible biological parallel for this sort of behaviour would be that of actin waves on the surface of cells (Gerisch et al. 2009). In order properly replicate this behaviour it would be necessary to create an additional constraint wherein the *total* concentration remains constant.

4.5 Radially-Symmetric 2D

It is possible to obtain a radially symmetric 2D system by replacing the diffusion terms in 4.1 with $\Delta = \frac{\partial^2}{\partial r^2} + \frac{1}{r} \frac{\partial}{\partial r}$, which results in a pseudo-2D system.

From here similar behaviours are observed as in the simple 1D-case.

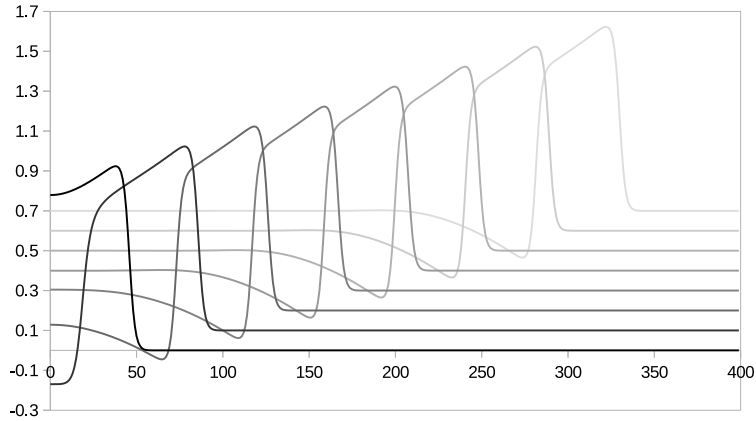


Figure 4.7: Propagating Wave in Radially-Symmetric 2D FitzHugh-Nagumo System

$D_v = 1$. Profiles for greater values of time are shown in paler shades of grey.

Again, as in the 1D case for $D_v = 1$ we have a wave which propagates outwards from the centre of the medium.

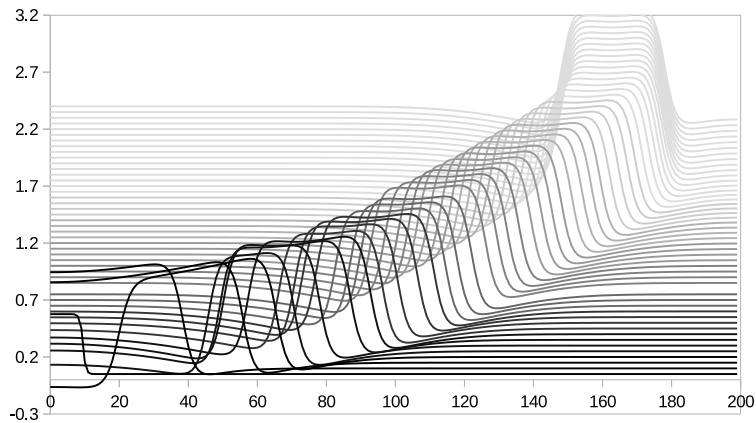


Figure 4.8: Stationary Annulus in Radially-Symmetric 2D FitzHugh-Nagumo System

$D_v = 4$. Profiles for greater values of time are shown in paler shades of grey.

The stationary annulus occurs because of the interaction with the boundary; chemically, the annulus is being forced outwards, but the boundary repels it.

Under certain conditions on ϵ_v and D_v , when D_v is sufficiently large as in the 1D case and the excitation does not split into a ring, a stationary spot in the centre of the medium can also be observed. This is more similar to the stationary spot observed in the 1D system.

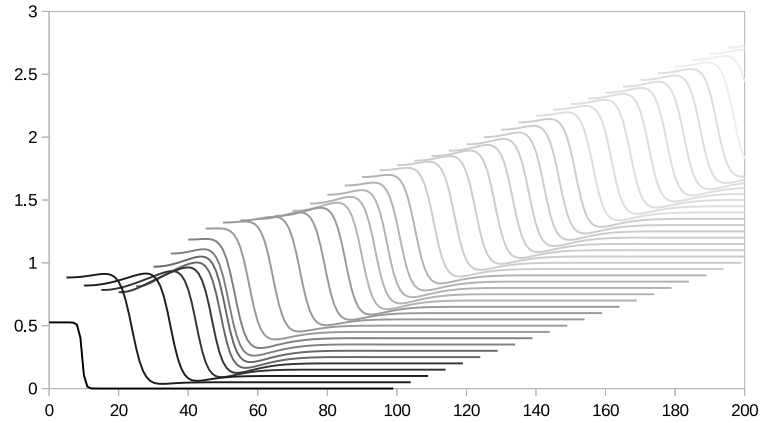


Figure 4.9: Stationary spot in Radially-Symmetric 2D FitzHugh-Nagumo System

$D_v = 2.8$, $\epsilon_v = 0.25$. Profiles for greater values of time are shown in paler shades of grey.

Crucially, this is a stationary pattern without interaction with the boundaries.

In reality, systems are unlikely to be radially-symmetric and so a more realistic full-2D system should be considered.

4.6 Fitzhugh-Nagumo in 2-dimensions

A natural progression from the one-dimensional system is to expand it to two-dimensions.

With varying diffusion coefficients, similar sorts of behaviours are observed as in the 1D and radially-symmetric case. $D_v = 1$ gives travelling waves in the form of an expanding annulus.

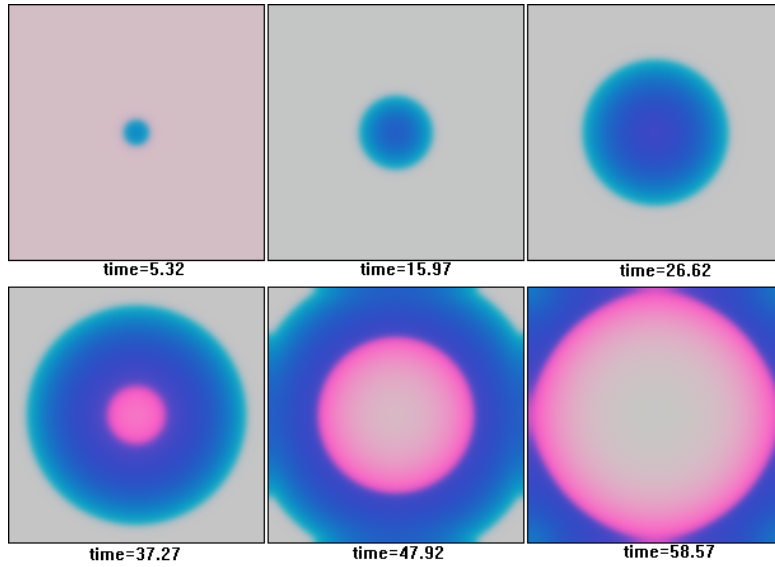


Figure 4.10: Two-dimensional FitzHugh-Nagumo Model - Travelling Wave
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 1$, which results in, as in the 1D-case, a travelling wave.

As $D_v \rightarrow 3$ the patterns develop oscillatory behaviour, though the patterns are ultimately unstable, returning to the homogenous state eventually.

At $D_v = 3$, the patterns oscillate stably.

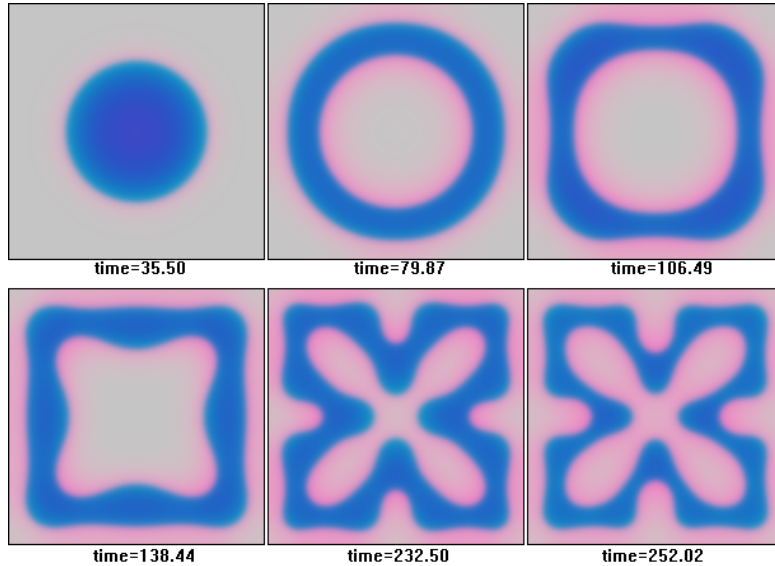


Figure 4.11: Two-dimensional FitzHugh-Nagumo Model - Oscillating Pattern

Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 3$, which results in a pattern that oscillates between the fifth and sixth frames.

Then for $D_v = 4$ stationary patterns are observed.

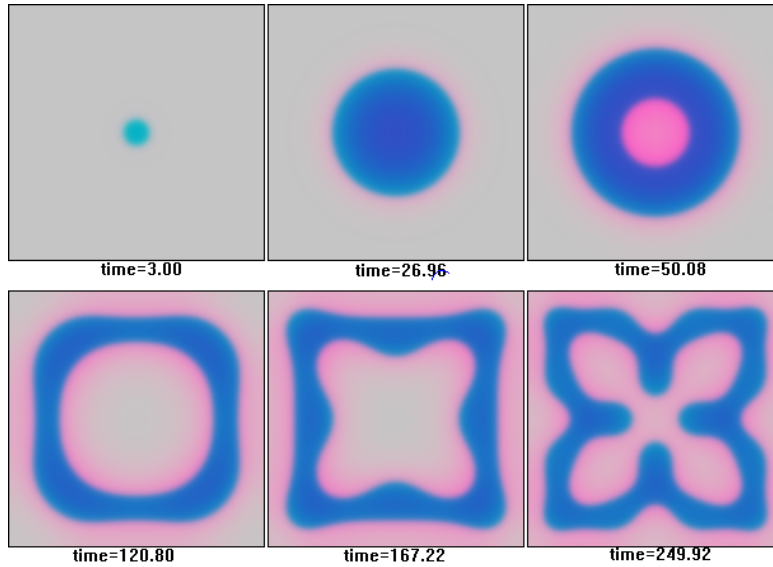


Figure 4.12: Two-dimensional FitzHugh-Nagumo Model - Stationary Pattern

Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200×200 . Here $D_v = 4$, which results in, as in the 1D-case, a stationary pattern. However, interaction with the boundaries results in a stationary pattern which is not a simple spot. Notice that the activator appears to repel itself.

In both 4.11 and 4.12, the patterns are stable due to interaction with the boundary, as in the radially-symmetric stationary annulus. However, due to angular differences, the curvature of these patterns is not uniform.

Again, as in the radially symmetric case, under certain conditions (i.e. sufficiently large values of D_v and the excitation not splitting into a ring), stationary spots can form without interaction with the boundary.

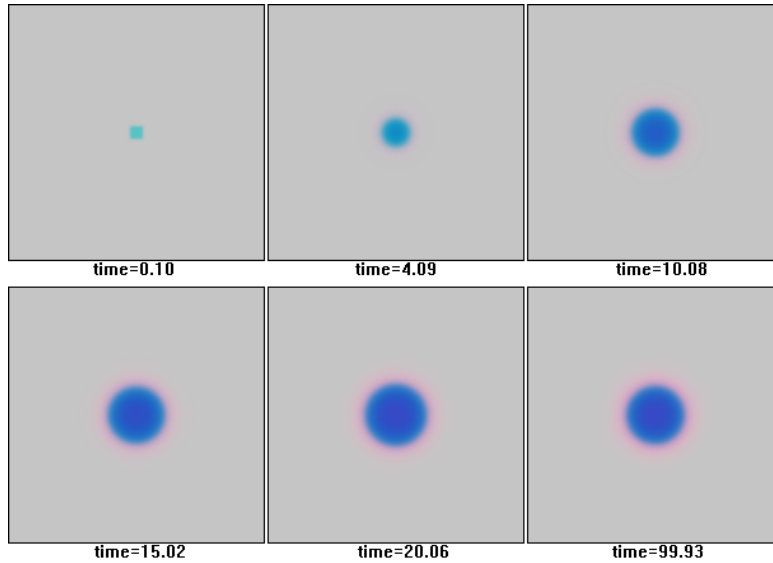


Figure 4.13: Two-dimensional FitzHugh-Nagumo Model - Stationary Spot
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200×200 . Here $D_v = 2.8$, $\epsilon_v = 0.25$ which results in a stationary spot in the centre without interacting with the boundaries of the medium.

Here the spot expands and rather than collapsing into a ring as in the previous case, it reaches some fixed size and becomes stable. This is more similar to the 1D stationary spot.

4.7 Noise in the 2D FitzHugh-Nagumo System

Applying noise to the 2D system as in 4.4 we can obtain a better approximation for the behaviour observed in (Gerisch et al. 2009), since we are considering excitations on the surface of a cell.

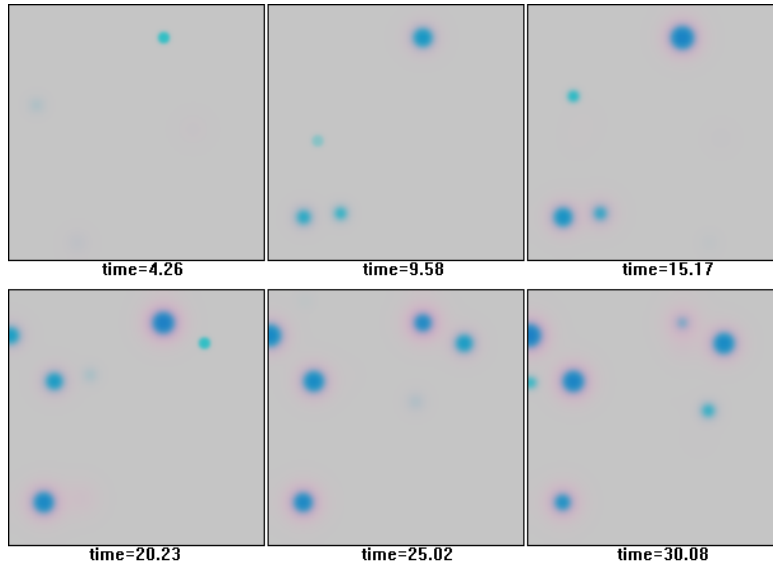


Figure 4.14: Two-dimensional FitzHugh-Nagumo Model with noise
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 2$, $u_0 = 0.15$, $\epsilon_v = 0.2$. Again, small perturbations are applied every 500 timesteps.

As in the 1D case, for sufficiently large excitation, spots appear, grow and remain for a short time before disappearing. For smaller excitations, they very quickly disappear. For a clearer illustration of this behaviour, see the videos referenced in the appendices.

Chapter 5

Conclusion

During the course of this project I have not only accurately reproduced earlier results, but also obtained some new results.

I have studied the general theory of pattern formation, including studying Michaelis-Menten kinetics as a simple example of a biochemical reaction as well as some simple examples of oscillatory, excitable and multi-stable systems.

I then studied scaling in a 1D diffusion-decay system, reproducing previously known results as well as developing ideas regarding the asymptotic behavior of scaling under Dirichlet and Neumann boundary conditions.

Futhermore I have developed my own computer programs in C++ to simulate reaction-diffusion systems, in particular a modified Fitzhugh-Nagumo model, using numerical integration methods to achieve this. The simulations were conducted in a number of spatial systems: a simple 1D case, a radially symmetric 2D case and a full 2D case. The simulations consisted of making small perturbations to the centre of a medium with otherwise uniform concentrations of two chemicals, the activator u and inhibitor v . Amongst the behaviours observed were travelling waves, pulsating and stationary spots.

In addition I have used analytic techniques to study the stability of one particular class of patterns observed in the 1D system and thus finding conditions on the parameters in order for stability to be achieved.

Finally, I also simulated the addition of random noise in both the 1D and 2D systems, periodically applying small random perturbations to an otherwise homogeneous medium in random locations in order to model an observed biological behaviour. These simulations modelled the appearance of excited areas on the surface of a cell which appear randomly, some of which disappear rapidly, others growing in amplitude and remaining for a moment before disappearing depending on whether the random perturbations were sufficiently large.

Bibliography

- Edelstein-Keshet, Leah (1988). *Mathematical Models in Biology*. Society for Industrial and Applied Mathematics. ISBN: 0-89871-554-7.
- Gerisch, Gunther et al. (2009). “Self-organizing actin waves as planar phagocytic cup structures”. In: *Cell Adhesion & Migration* 3:4, pp. 373–382.
- Ishihara, Shuji and Kunihiko Kaneko (2006). “Turing pattern with proportion preservation”. In: *Journal of Theoretical Biology* 238, pp. 683–693.
- Meinhardt, Hans (2008). “Models of biological pattern formation: from elementary steps to the organization of embryonic axes”. In: *Current Topics in Developmental Biology* 81, pp. 1–63.
- (2009). *The Algorithmic Beauty of Sea Shells 4th Ed.* Springer. ISBN: 978-3-540-92141-7.
- Murray, James D. (1989). *Mathematical Biology*. Springer. ISBN: 3-540-57204-X.
- Rasolonjanahary, Manan’Iarivo (2013). “Scaling of morphogenetic patterns in continuous and discrete models”. PhD thesis. University of Liverpool.
- Turing, Alan M. (1952). “The Chemical Basis for Morphogenesis”. In: *Philosophical Transactions of the Royal Society of London* 237.641, pp. 37–72.
- Vasiev, Bakhtier (2004). “Classification of patterns in excitable systems with lateral inhibition”. In: *Physics Letters A* 323, pp. 192–203.

Appendices

Videos of Simulations

Videos of all the simulations depicted in chapter 4 are available at <http://pcwww.liv.ac.uk/~sgaphill/> (unfortunately only visible via University-networked computers).

Source Code

Abbreviated versions of the source code for the programs generating the simulations depicted in chapter 4 are printed here. For conciseness, not included is the header files, standard MFC boilerplate code and a very long function to output frames to a tif file. Full versions of the source code, including a Visual Studio 2008 solution, are available at: <http://pcwww.liv.ac.uk/~sgaphill/> (unfortunately only visible via University-networked computers).

One-dimensional Fitzhugh-Nagumo Model

```
1 // 1DFHN.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "ReactionDiffusion.h"
6 #include "ReactionDiffusionDlg.h"
7 #include <math.h>
8 #include <afxwin.h>
9
10 #ifdef _DEBUG
11 #define new DEBUG_NEW
12 #endif
13
14 void CReactionDiffusionDlg::OnPaint()
15 {
16     CPaintDC olddc(this); // device context for painting
17     CDC dc; //We will bitblit this to olddc
18     CBitmap bmpdc;
19
20     if (IsIconic())
21     {
22         ...
23     }
24     else
25     {
26         CDialog::OnPaint();
```

```

27
28     //Get plot area and save dimensions
29     CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
30     CRect R; pWnd->GetWindowRect(&R);
31     INT nPlotWidth = R.Width();
32     INT nPlotHeight = R.Height();
33
34     ScreenToClient(&R);
35     CPoint r,s;
36     CPoint origin;
37     LPCRECT plot_area(R);
38
39     dc.CreateCompatibleDC(&dc);
40     bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
Height());
41     dc.SelectObject(&bmpdc);
42
43     double xi,xx1,xx2,yy1,yy2,x_unit,y_unit;
44     CString str;
45
46     const int MEDIUM_SIZE = 200; //Size of medium
47
48     bool dump = false; //Dump each frame to a tif file?
49
50     const double Du = 1.0; //Diffusion coefficient of u
51     double Dv = 1.0; //Diffusion coefficient of v
52     double eu = 1.0; //Rate of kinetics of u
53     double ev = 0.1; //Rate of kinetics of v
54     double ku = 4.5;
55     double u1 = 1;
56     double u0 = 0.05;
57
58     double ju0 = 0.0; //boundary flux of u at i=0
59     double ju1 = 0.0; //boundary flux of u at i=
MEDIUM_SIZE-1
60     double jv0 = 0.0; //boundary flux of v at i=0
61     double jv1 = 0.0; //boundary flux of v at i=
MEDIUM_SIZE-1
62
63     double u[MEDIUM_SIZE];
64     double v[MEDIUM_SIZE];

```

```

65
66     //Arrays for storing concentrations at next time
step
67     double u_next[MEDIUM_SIZE];
68     double v_next[MEDIUM_SIZE];
69
70     double delta_x=0.4; //space step
71     double delta_x_square = delta_x*delta_x; //squaring
to save computation
72     double delta_t; //time step
73
74     //Define delta_t in relation to Dv to ensure
stability
75     if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
76     else delta_t=0.005;
77
78     double current_time = 0.0;
79     double final_time = 300;
80
81     int current_step=0;
82     int draw_interval=1; //Interval between drawing
steps
83     int offset = 0; //First timestep to draw
84
85     //Index for for-loops
86     int i;
87
88     //Number of segments on each axis
89     int x_segments=10;
90     int y_segments=10;
91
92     //Min and max for y axis
93     double y_max = 1;
94     double y_min = -0.2;
95
96     //Interval between segments
97     double x_interval = MEDIUM_SIZE/x_segments;
98     double y_interval = (y_max-y_min)/y_segments;
99
100    //Du,Dv divided by delta_x^2 to save time
101    double Du_div = Du/(delta_x*delta_x);

```

```

102         double Dv_div = Dv/(delta_x*delta_x);
103
104         //Setting up initial conditions
105         for (i=0;i<MEDIUM_SIZE;i++)
106         {
107             u[i]=0.1;
108             v[i]=0.1;
109         }
110
111         for (i=MEDIUM_SIZE/2 - 5;i<MEDIUM_SIZE/2 + 5;i++)
112         {
113             u[i]=0.5;
114         }
115
116         while(current_time<final_time)
117         {
118             //Flux at boundaries
119             u[0] = u[1] + ju0*delta_x;
120             v[0] = v[1] + jv0*delta_x;
121             u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*
delta_x;
122             v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*
delta_x;
123
124             //Explicit Euler method
125             for(i=1;i<MEDIUM_SIZE-1;i++)
126             {
127                 u_next[i] = u[i] + delta_t*(Du_div*(u[i
+1]-2*u[i]+u[i-1]) - eu*(ku*u[i]*(u[i]-u0)*(u[i]-u1) + v[
i]));
128                 v_next[i] = v[i] + delta_t*(Dv_div*(v[i
+1]-2*v[i]+v[i-1]) + ev*(u[i] - v[i]));
129             }
130
131             //Draw curves every "draw_interval"-steps
132             if((current_step-offset)%draw_interval==0)
133             {
134
135                 //Clear plot area
136                 dc.FillRect(plot_area,WHITE_BRUSH);
137

```

```

138      //Select blackPen
139      dc.SelectObject(blackPen);
140
141      xx1=R.TopLeft().x + 0.05*R.Width();
142      yy1=R.BottomRight().y - 0.05*R.Height();
143      xx2=R.BottomRight().x - 0.05*R.Width();
144      yy2=R.TopLeft().y + 0.05*R.Height();
145
146      double x_unit = (xx2-xx1)/MEDIUM_SIZE;
147      double y_unit = (yy2-yy1)/(y_max-y_min);
148
149      origin.x = xx1;
150      origin.y = yy1 - y_min*y_unit;
151
152      //Draw x-axis
153      if(y_min>=0)
154      {
155          dc.MoveTo(xx1,yy1);
156          dc.LineTo(xx2,yy1);
157      }
158      else
159      {
160          dc.MoveTo(xx1,origin.y);
161          dc.LineTo(xx2,origin.y);
162      }
163
164      //Draw y-axis
165      dc.MoveTo(xx1,yy1);
166      dc.LineTo(xx1,yy2);
167
168      //Label x-axis
169      if(y_min>=0)
170      {
171          s.x=xx1;
172          s.y=yy1;
173      }
174      else
175      {
176          s.x=origin.x;
177          s.y=origin.y;
178      }

```

```

179         for(i=0;i<=x_segments;i++)
180         {
181             r.y = s.y - 0.01*R.Height();
182             r.x = s.x + i*(xx2-xx1)/x_segments;
183             dc.MoveTo(r.x,r.y);
184             r.y = s.y + 0.01*R.Height();
185             dc.LineTo(r.x,r.y);
186             str.Format(_T("%0.f"),x_interval*i);
187             dc.TextOut(r.x-10,r.y+1,str);
188         }
189
190         //Label y-axis
191         for(i=0;i<=y_segments;i++)
192         {
193             r.x = xx1 - 0.01*R.Height();
194             r.y = yy1 + i*(yy2-yy1)/y_segments;
195             dc.MoveTo(r.x,r.y);
196             r.x = xx1 + 0.01*R.Height();
197             dc.LineTo(r.x,r.y);
198             str.Format(_T("%0.2f"),y_min +
199 y_interval*i);
200             dc.TextOut(r.x-50,r.y-7,str);
201         }
202
203         str.Format(_T("time=%0.2f"),current_time);
204         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
205 +30,str);
206
207         str.Format(_T("activator_ _blue"));
208         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
209 +60,str);
210
211         str.Format(_T("inhibitor_ _red"));
212         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
213 +75,str);
214
215         //Draw curve for u
216         dc.SelectObject(bluePen);
217         xi=(float)0;
218         r.x=origin.x+xi*x_unit;
219         r.y=origin.y+u[0]*y_unit;
220         dc.MoveTo(r.x,r.y);

```



```

216         for (i=0;i<MEDIUM_SIZE-1;i++)
217         {
218             xi=(float)i;
219             r.x=origin.x+xi*x_unit;
220             r.y=origin.y+u[i]*y_unit;
221             dc.LineTo(r.x,r.y);
222         }
223
224         //Draw curve for v
225         dc.SelectObject(redPen);
226         xi=(float)0;
227         r.x=origin.x+xi*x_unit;
228         r.y=origin.y+v[0]*y_unit;
229         dc.MoveTo(r.x,r.y);
230
231         for (i=1;i<MEDIUM_SIZE;i++)
232         {
233             xi=(float)i;
234             r.x=origin.x+xi*x_unit;
235             r.y=origin.y+v[i]*y_unit;
236             dc.LineTo(r.x,r.y);
237         }
238
239         //Bit-blit to display
240         olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(), &dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
241         //Dump frame to tif file
242         if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x, R.TopLeft().y,&dc, current_step/
draw_interval);
243
244         //Dump values to log: Tab-separated values
245         //                               Time steps on separate
lines
246         FILE * pFile;
247         char fname[32];
248         sprintf(fname,"log\\log.txt");
249
250         if((current_step-offset)/draw_interval == 0)
pFile = fopen (fname, "wb");

```

```

251         else pFile = fopen (fname, "ab");
252
253         for (i=0;i<MEDIUM_SIZE;i++)
254         {
255             fprintf (pFile, "%f\t",u[i]);
256         }
257         fprintf (pFile, "\n");
258         //for (i=0;i<MEDIUM_SIZE;i++)
259         //{
260         //    fprintf (pFile, "%f\t",v[i]);
261         //}
262         //fprintf (pFile, "\n");
263         fclose (pFile);
264     }
265
266     for (i=0;i<MEDIUM_SIZE;i++)
267     {
268         u[i]=u_next[i];
269         v[i]=v_next[i];
270     }
271
272     //Increment
273     current_time+=delta_t;
274     current_step++;
275 }
276 }
277 }
278
279 BOOL CReactionDiffusionDlg::DumpToFile(int width, int height
    , int istart, int jstart, CDC* dc, int count)
280 {
281     ...
282 }

```

One-dimensional Fitzhugh-Nagumo Model with noise

```

1 // 1DFHN-with-noise.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "ReactionDiffusion.h"

```

```

6 #include "ReactionDiffusionDlg.h"
7 #include <math.h>
8 #include <afxwin.h>
9 #include <stdlib.h>
10
11 #ifdef _DEBUG
12 #define new DEBUG_NEW
13 #endif
14
15 void CReactionDiffusionDlg::OnPaint()
16 {
17     CPaintDC olddc(this); // device context for painting
18     CDC dc; //We will bitblit this to olddc
19     CBitmap bmpdc;
20
21     if (IsIconic())
22     {
23         ...
24     }
25     else
26     {
27         CDialog::OnPaint();
28
29         //Get plot area and save dimensions
30         CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
31         CRect R; pWnd->GetWindowRect(&R);
32         INT nPlotWidth = R.Width();
33         INT nPlotHeight = R.Height();
34
35         ScreenToClient(&R);
36         CPoint r,s;
37         CPoint origin;
38         LPCRECT plot_area(R);
39
40         dc.CreateCompatibleDC(&dc);
41         bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
Height());
42         dc.SelectObject(&bmpdc);
43
44         double xi,xx1,xx2,yy1,yy2,x_unit,y_unit;
45         CString str;

```

```

46
47     const int MEDIUM_SIZE = 200; //Size of medium
48
49     bool dump = false; //Dump each frame to a tif file?
50
51     const double Du = 1.0; //Diffusion coefficient of u
52     double Dv = 2.0; //Diffusion coefficient of v
53     double eu = 1.0; //Rate of kinetics of u
54     double ev = 0.2; //Rate of kinetics of v
55     double ku = 4.5;
56     double u1 = 1;
57     double u0 = 0.15;
58
59     double stim,a,init;
60
61     //Seed for RNG
62     int seed = 123;
63
64     double ju0 = 0.0; //boundary flux of u at i=0
65     double ju1 = 0.0; //boundary flux of u at i=
66     MEDIUM_SIZE-1
67     double jv0 = 0.0; //boundary flux of v at i=0
68     double jv1 = 0.0; //boundary flux of v at i=
69     MEDIUM_SIZE-1
70
71     double u[MEDIUM_SIZE];
72     double v[MEDIUM_SIZE];
73
74     double u_next[MEDIUM_SIZE];
75     double v_next[MEDIUM_SIZE];
76
77     double delta_x=0.4; //space step
78     double delta_x_square = delta_x*delta_x; //squaring
79     to save computation later
80     double delta_t; //time step
81
82     //Define delta_t in relation to Dv to ensure
83     stability
84     if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
85     else delta_t=0.005;

```

```

83
84     double current_time = 0.0;
85     double final_time = 300;
86
87     int current_step=0;
88     int draw_interval=1; //Interval between drawing
steps
89     int offset = 0; //First timestep to draw
90
91     //Index for for-loops
92     int i;
93
94     //Number of segments on each axis
95     int x_segments=10;
96     int y_segments=10;
97
98     //Min and max for y axis
99     double y_max = 1;
100    double y_min = -0.2;
101
102    //Interval between segments
103    double x_interval = MEDIUM_SIZE/x_segments;
104    double y_interval = (y_max-y_min)/y_segments;
105
106    //Du, Dv divided by delta_x^2 to save time
107    double Du_div = Du/(delta_x*delta_x);
108    double Dv_div = Dv/(delta_x*delta_x);
109
110    //Setting up initial conditions
111    for (i=0;i<MEDIUM_SIZE;i++)
112    {
113        u[i]=0.1;
114        v[i]=0.1;
115    }
116
117    srand(seed);
118    a=0;
119
120
121    while(current_time<final_time)
122    {

```

```

123         //Flux at boundaries
124         u[0] = u[1] + ju0*delta_x;
125         v[0] = v[1] + jv0*delta_x;
126         u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*
delta_x;
127         v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*
delta_x;
128         stim = 0;
129
130         //Apply random noise at random position every
100 timesteps
131         if(current_step%500==0)
132         {
133             init = (double) rand();
134             init = init/RAND_MAX;
135             init = init*MEDIUM_SIZE;
136
137             a = (double) rand();
138             a = a/RAND_MAX;
139             a = a - 0.5;
140         }
141
142         //Explicit Euler method
143         for(i=1;i<MEDIUM_SIZE-1;i++)
144         {
145             if(current_step%500<100 && i>init-5 && i<
init+5)
146             {
147                 stim = a*0.4;
148             }
149             else
150             {
151                 stim = 0;
152             }
153
154             u_next[i] = u[i] + delta_t*(Du_div*(u[i
+1]-2*u[i]+u[i-1]) - eu*(ku*u[i]*(u[i]-u0)*(u[i]-u1) + v[
i]) + stim);
155             v_next[i] = v[i] + delta_t*(Dv_div*(v[i
+1]-2*v[i]+v[i-1]) + ev*(u[i] - v[i]));
156         }

```

```

157
158 //Draw curves every "draw_interval"-steps
159 if((current_step-offset)%draw_interval==0)
160 {
161     //Clear plot area
162     dc.FillRect(plot_area,WHITE_BRUSH);
163     //Select blackPen
164     dc.SelectObject(blackPen);
165
166     xx1=R.TopLeft().x + 0.05*R.Width();
167     yy1=R.BottomRight().y - 0.1*R.Height();
168     xx2=R.BottomRight().x - 0.05*R.Width();
169     yy2=R.TopLeft().y + 0.05*R.Height();
170
171     double x_unit = (xx2-xx1)/MEDIUM_SIZE;
172     double y_unit = (yy2-yy1)/(y_max-y_min);
173
174     origin.x = xx1;
175     origin.y = yy1 - y_min*y_unit;
176
177     //Draw x-axis
178     if(y_min>=0)
179     {
180         dc.MoveTo(xx1,yy1);
181         dc.LineTo(xx2,yy1);
182     }else
183     {
184         dc.MoveTo(xx1,origin.y);
185         dc.LineTo(xx2,origin.y);
186     }
187
188     //Draw y-axis
189     dc.MoveTo(xx1,yy1);
190     dc.LineTo(xx1,yy2);
191
192     //Label x-axis
193     if(y_min>=0)
194     {
195         s.x=xx1;
196         s.y=yy1;
197     }

```

```

198         else
199         {
200             s.x=origin.x;
201             s.y=origin.y;
202         }
203         for(i=0;i<=x_segments;i++)
204         {
205             r.y = s.y - 0.01*R.Height();
206             r.x = s.x + i*(xx2-xx1)/x_segments;
207             dc.MoveTo(r.x,r.y);
208             r.y = s.y + 0.01*R.Height();
209             dc.LineTo(r.x,r.y);
210             str.Format(_T("%0.2f"),x_interval*i);
211             dc.TextOut(r.x-10,r.y+1,str);
212         }
213
214         //Label y-axis
215         for(i=0;i<=y_segments;i++)
216         {
217             r.x = xx1 - 0.01*R.Height();
218             r.y = yy1 + i*(yy2-yy1)/y_segments;
219             dc.MoveTo(r.x,r.y);
220             r.x = xx1 + 0.01*R.Height();
221             dc.LineTo(r.x,r.y);
222             str.Format(_T("%0.2f"),y_min +
y_interval*i);
223             dc.TextOut(r.x-50,r.y-7,str);
224         }
225
226         str.Format(_T("time=%0.2f"),current_time);
227         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
+30,str);
228         str.Format(_T("activator_□_□blue"));
229         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
+60,str);
230         str.Format(_T("inhibitor_□_□red"));
231         dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
+75,str);
232
233         //Draw curve for u
234         dc.SelectObject(bluePen);

```



```

235         xi=(float)0;
236         r.x=origin.x+xi*x_unit;
237         r.y=origin.y+u[0]*y_unit;
238         dc.MoveTo(r.x,r.y);
239
240         for (i=0;i<MEDIUM_SIZE-1;i++)
241         {
242             xi=(float)i;
243             r.x=origin.x+xi*x_unit;
244             r.y=origin.y+u[i]*y_unit;
245             dc.LineTo(r.x,r.y);
246         }
247
248         //Draw curve for v
249         dc.SelectObject(redPen);
250         xi=(float)0;
251         r.x=origin.x+xi*x_unit;
252         r.y=origin.y+v[0]*y_unit;
253         dc.MoveTo(r.x,r.y);
254
255         for (i=1;i<MEDIUM_SIZE;i++)
256         {
257             xi=(float)i;
258             r.x=origin.x+xi*x_unit;
259             r.y=origin.y+v[i]*y_unit;
260             dc.LineTo(r.x,r.y);
261         }
262
263
264         olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
265         //Dump frame to tif file
266         if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
;
267
268         //Dump values to log: Tab-separated values
269         //                               Time steps on separate
lines
270         FILE * pFile;

```

```

271         char fname[32];
272         sprintf(fname,"log\\log.txt");
273
274         if((current_step-offset)/draw_interval == 0)
pFile = fopen (fname, "wb");
275         else pFile = fopen (fname, "ab");
276
277         for (i=0;i<MEDIUM_SIZE;i++)
278         {
279             fprintf (pFile, "%f\t",u[i]);
280         }
281         fprintf (pFile, "\n");
282         //for (i=0;i<MEDIUM_SIZE;i++)
283         //{
284             //    fprintf (pFile, "%f\t",v[i]);
285         //}
286         //fprintf (pFile, "\n");
287         fclose (pFile);
288     }
289
290     for (i=0;i<MEDIUM_SIZE;i++)
291     {
292         u[i]=u_next[i];
293         v[i]=v_next[i];
294     }
295
296     //Increment
297     current_time+=delta_t;
298     current_step++;
299 }
300 }
301 }
302
303 BOOL CReactionDiffusionDlg::DumpToFile(int width, int height
    , int istart, int jstart, CDC* dc, int count)
304 {
305     ...
306 }

```

Radially-Symmetric Fitzhugh-Nagumo Model

```

1 // Radially-Symmetric-FHN.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "ReactionDiffusion.h"
6 #include "ReactionDiffusionDlg.h"
7 #include <math.h>
8 #include <afxwin.h>
9
10 #ifdef _DEBUG
11 #define new DEBUG_NEW
12 #endif
13
14 void CReactionDiffusionDlg::OnPaint()
15 {
16     CPaintDC olddc(this); // device context for painting
17     CDC dc; //We will bitblit this to olddc
18     CBitmap bmpdc;
19
20     if (IsIconic())
21     {
22         ...
23     }
24     else
25     {
26         CDialog::OnPaint();
27
28         //Get plot area and save dimensions
29         CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
30         CRect R; pWnd->GetWindowRect(&R);
31         INT nPlotWidth = R.Width();
32         INT nPlotHeight = R.Height();
33
34         ScreenToClient(&R);
35         CPoint r,s,plot_origin,graph_origin;
36         LPCRECT plot_area(R);
37
38         COLORREF pixel_colour = RGB(0,0,0);
39
40         dc.CreateCompatibleDC(&dc);

```

```

41     bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
Height());
42     dc.SelectObject(&bmpdc);
43
44     double plot_left,plot_right,graph_left,graph_right,
top,bottom,x_unit,y_unit,border,xj;
45     double activator,inhibitor,zero_level;
46     CString str;
47
48     const int MEDIUM_SIZE = 200; //Size of medium
49
50     //Minimizing computation for later plotting of
medium
51     int M2 = 2*MEDIUM_SIZE;
52     int Msq = MEDIUM_SIZE*MEDIUM_SIZE;
53
54     bool dump = false; //Dump each frame to a tif file?
55
56     const double Du = 1.0; //Diffusion coefficient of u
57     double Dv = 1.0; //Diffusion coefficient of v
58     double eu = 1.0; //Rate of kinetics of u
59     double ev = 0.1; //Rate of kinetics of v
60     double ku = 4.5;
61     double u1 = 1;
62     double u0 = 0.05;
63
64     double ju0 = 0.0; //boundary flux of u at i=0
65     double ju1 = 0.0; //boundary flux of u at i=
MEDIUM_SIZE-1
66     double jv0 = 0.0; //boundary flux of v at i=0
67     double jv1 = 0.0; //boundary flux of v at i=
MEDIUM_SIZE-1
68
69     double u[MEDIUM_SIZE];
70     double v[MEDIUM_SIZE];
71
72     //Arrays for storing concentrations at next time
step
73     double u_next[MEDIUM_SIZE];
74     double v_next[MEDIUM_SIZE];
75

```

```

76     double delta_x=0.4; //space step
77     double delta_x_square = delta_x*delta_x; //squaring
    to save computation later
78     double delta_t; //time step
79
80     //Define delta_t in relation to Dv to ensure
    stability
81     if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
82     else delta_t=0.005;
83
84     double current_time = 0.0;
85     double final_time = 500;
86
87     int current_step=0;
88     int draw_interval=10; //Interval between drawing
    steps
89     int offset = 0; //First timestep to draw
90
91     //Index for for-loops
92     int i,j,ii,isq,distsq;
93
94     //Number of segments on each axis
95     int x_segments=10;
96     int y_segments=10;
97     double y_max = 1;
98     double y_min = -0.3;
99
100    //Interval between segments
101    double x_interval = MEDIUM_SIZE/x_segments;
102    double y_interval = (y_max-y_min)/y_segments;
103
104    border = 0.05*min(R.Height(),R.Width());
105    top = R.TopLeft().y + border;
106    bottom = R.BottomRight().y - border;
107    plot_left = R.TopLeft().x + border;
108    plot_right = R.TopLeft().x + 0.5*R.Width() - border;
109    graph_right = R.BottomRight().x - border;
110    graph_left = R.TopLeft().x + 0.5*R.Width() + border;
111    plot_origin.x = int(floor((plot_right - plot_left -
    2*MEDIUM_SIZE)*0.5));

```

```

112     plot_origin.y = int(floor((bottom - top - 2*
MEDIUM_SIZE)*0.5));
113
114     x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
115     y_unit = (top-bottom)/(y_max-y_min);
116     graph_origin.x = graph_left;
117     graph_origin.y = bottom - y_min*y_unit;
118
119     //Du,Dv divided by delta_x^2 to save time
120     double Du_div = Du/(delta_x*delta_x);
121     double Dv_div = Dv/(delta_x*delta_x);
122
123     //Setting up initial conditions
124     for (i=0;i<MEDIUM_SIZE;i++)
125     {
126         u[i]=0.1;
127         v[i]=0.1;
128     }
129
130     //Perturb at centre of medium
131     for (i=0;i<5;i++)
132     {
133         u[i]=0.5;
134     }
135
136     while(current_time<final_time)
137     {
138
139         //Flux at boundaries
140         u[0] = u[1] + ju0*delta_x;
141         v[0] = v[1] + jv0*delta_x;
142         u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*
delta_x;
143         v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*
delta_x;
144
145         //Explicit Euler method
146         for(i=1;i<MEDIUM_SIZE-1;i++)
147         {
148             u_next[i] = u[i] + delta_t*(Du_div*(u[i
+1]-2*u[i]+u[i-1] +(u[i+1]-u[i])/i) - eu*(ku*u[i]*(u[i]-

```

```

u0)*(u[i]-u1) + v[i]));
149         v_next[i] = v[i] + delta_t*(Dv_div*(v[i
+1]-2*v[i]+v[i-1] +(v[i+1]-v[i])/i) + ev*(u[i] - v[i]));
150     }
151
152     //Draw curves every "draw_interval"-steps
153     if((current_step-offset)%draw_interval==0)
154     {
155         y_interval = (y_max-y_min)/y_segments;
156         y_unit = (top-bottom)/(y_max-y_min);
157
158         //Clear plot area
159         dc.FillRect(plot_area,WHITE_BRUSH); //Using
WHITE_BRUSH proper?
160
161         //Plot on left
162         //Draw bounding box
163         dc.SelectObject(blackPen);
164         dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
165         dc.LineTo(plot_origin.x+2*MEDIUM_SIZE,
plot_origin.y-1);
166         dc.LineTo(plot_origin.x+2*MEDIUM_SIZE,
plot_origin.y+2*MEDIUM_SIZE);
167         dc.LineTo(plot_origin.x-1,plot_origin.y+2*
MEDIUM_SIZE);
168         dc.LineTo(plot_origin.x-1,plot_origin.y-1);
169
170         for(i=0;i<M2;i++)
171         {
172             ii=i*M2;
173             isq = (i-MEDIUM_SIZE)*(i-MEDIUM_SIZE);
174             for(j=0;j<M2;j++)
175             {
176                 distsq = isq + (j-MEDIUM_SIZE)*(j-
MEDIUM_SIZE);
177                 if(distsq <= Msq)
178                 {
179                     int loc = (int) floor(sqrt( (
double)distsq ));
180                     activator = u[loc];
181                     inhibitor = v[loc];

```

```

182         activator = int((activator-y_min
183 )*255/(y_max-y_min));
184         inhibitor = int((inhibitor-y_min
185 )*255/(y_max-y_min));
186         zero_level = int((0-y_min)*255/(
187 y_max-y_min));
188         pixel_colour = RGB(255-activator
189 ,255-inhibitor,255-zero_level);
190     }
191     else
192     {
193         pixel_colour = RGB(0,0,0);
194     }
195     dc.SetPixel(plot_origin.x+j,
196 plot_origin.y+i, pixel_colour);
197     }
198 }
199
200 //Plot graph of cross section at right
201 //Select blackPen
202 dc.SelectObject(blackPen);
203
204 //Draw x-axis
205 if(y_min>=0)
206 {
207     dc.MoveTo(graph_left,bottom);
208     dc.LineTo(graph_right,bottom);
209 }
210 else
211 {
212     dc.MoveTo(graph_left,graph_origin.y);
213     dc.LineTo(graph_right,graph_origin.y);
214 }
215
216 //Draw y-axis
217 dc.MoveTo(graph_left,bottom);
218 dc.LineTo(graph_left,top);
219
220 //Label x-axis
221 if(y_min>=0)
222 {

```



```

218         s.x=graph_origin.x;
219         s.y=bottom;
220     }
221     else
222     {
223         s.x=graph_origin.x;
224         s.y=graph_origin.y;
225     }
226     for(j=0;j<=x_segments;j++)
227     {
228         r.y = s.y - 0.01*R.Height();
229         r.x = s.x + j*(graph_right-graph_left)/
x_segments;
230         dc.MoveTo(r.x,r.y);
231         r.y = s.y + 0.01*R.Height();
232         dc.LineTo(r.x,r.y);
233         str.Format(_T("%0.2f"),x_interval*j);
234         dc.TextOut(r.x-10,r.y+1,str);
235     }
236
237     //Label y-axis
238     for(j=0;j<=y_segments;j++)
239     {
240         r.x = graph_left - 0.01*R.Height();
241         r.y = bottom + j*(top-bottom)/y_segments
;
242         dc.MoveTo(r.x,r.y);
243         r.x = graph_left + 0.01*R.Height();
244         dc.LineTo(r.x,r.y);
245         str.Format(_T("%0.2f"),y_min +
y_interval*j);
246         dc.TextOut(r.x-50,r.y-7,str);
247     }
248
249     str.Format(_T("time=%0.2f"),current_time);
250     dc.TextOut(graph_left+70,top+30,str);
251     str.Format(_T("activator_□_□blue"));
252     dc.TextOut(graph_left+70,top+60,str);
253     str.Format(_T("inhibitor_□_□red"));
254     dc.TextOut(graph_left+70,top+75,str);
255

```

```

256         i=0;
257         dc.SelectObject(bluePen);
258         xj=(float)0;
259         r.x = graph_origin.x + xj*x_unit;
260         r.y = graph_origin.y + u[i]*y_unit;
261         dc.MoveTo(r.x,r.y);
262
263         for (i=1;i<MEDIUM_SIZE;i++)
264         {
265             xj=(float)i;
266             r.x=graph_origin.x+xj*x_unit;
267             r.y=graph_origin.y+u[i]*y_unit;
268             dc.LineTo(r.x,r.y);
269         }
270
271         i=0;
272         dc.SelectObject(redPen);
273         xj=(float)0;
274         r.x=graph_origin.x+xj*x_unit;
275         r.y=graph_origin.y+v[i]*y_unit;
276         dc.MoveTo(r.x,r.y);
277
278         for (i=i;i<MEDIUM_SIZE;i++)
279         {
280             xj=(float)i;
281             r.x=graph_origin.x+xj*x_unit;
282             r.y=graph_origin.y+v[i]*y_unit;
283             dc.LineTo(r.x,r.y);
284         }
285
286         olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
287         //Dump frame to tif file
288         if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
;
289
290         //Dump values to log: Tab-separated values
291         //                               Time steps on separate
lines

```

```

292         FILE * pFile;
293         char fname [32];
294         sprintf (fname, "log\\log.txt");
295
296         if ((current_step - offset) / draw_interval == 0)
pFile = fopen (fname, "wb");
297         else pFile = fopen (fname, "ab");
298
299         for (i=0; i<MEDIUM_SIZE; i++)
300         {
301             fprintf (pFile, "%f\t", u[i]);
302         }
303         fprintf (pFile, "\n");
304         //for (i=0; i<MEDIUM_SIZE; i++)
305         //{
306         //    fprintf (pFile, "%f\t", v[i]);
307         //}
308         //fprintf (pFile, "\n");
309         fclose (pFile);
310     }
311
312     for (i=0; i<MEDIUM_SIZE; i++)
313     {
314         u[i]=u_next[i];
315         v[i]=v_next[i];
316     }
317
318     //Increment
319     current_time+=delta_t;
320     current_step++;
321 }
322 }
323 }
324
325 BOOL CReactionDiffusionDlg::DumpToFile(int width, int height
    , int istart, int jstart, CDC* dc, int count)
326 {
327 ...
328 }

```

Two-dimensional Fitzhugh-Nagumo Model

```
1 // 2DFHN.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "ReactionDiffusion.h"
6 #include "ReactionDiffusionDlg.h"
7 #include <math.h>
8 #include <afxwin.h>
9 #include <stdlib.h>
10
11 #ifdef _DEBUG
12 #define new DEBUG_NEW
13 #endif
14
15 void CReactionDiffusionDlg::OnPaint()
16 {
17     CPaintDC olddc(this); // device context for painting
18     CDC dc; //We will bitblit this to olddc
19     CBitmap bmpdc;
20
21     if (IsIconic())
22     {
23         ...
24     }
25     else
26     {
27         CDialog::OnPaint();
28
29         //Get plot area and save dimensions
30         CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
31         CRect R; pWnd->GetWindowRect(&R);
32         INT nPlotWidth = R.Width();
33         INT nPlotHeight = R.Height();
34
35         ScreenToClient(&R);
36         CPoint r,s,plot_origin,graph_origin;
37         LPCRECT plot_area(R);
38
39         COLORREF pixel_colour = RGB(0,0,0);
40
```

```

41         dc.CreateCompatibleDC(&dc);
42         bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
Height());
43         dc.SelectObject(&bmpdc);
44
45         double plot_left, plot_right, graph_left, graph_right,
top, bottom, x_unit, y_unit, border, xj;
46         CString str;
47
48         const int MEDIUM_SIZE = 200; //Size of medium:
MEDIUM_SIZE * MEDIUM_SIZE
49
50         bool dump = false; //Dump each frame to a tif file?
51
52         const double Du = 1.0; //Diffusion coefficient of u
53         double Dv = 1.0; //Diffusion coefficient of v
54         double eu = 1.0; //Rate of kinetics of u
55         double ev = 0.1; //Rate of kinetics of v
56         double ku = 4.5;
57         double u1 = 1;
58         double u0 = 0.05;
59
60         double stim=0.;
61
62         double ju0 = 0.0; //boundary flux of u at i=0
63         double ju1 = 0.0; //boundary flux of u at i=
MEDIUM_SIZE-1
64
65         double jv0 = 0.0; //boundary flux of v at i=0
66         double jv1 = 0.0; //boundary flux of v at i=
MEDIUM_SIZE-1
67
68         int vector_size = MEDIUM_SIZE*MEDIUM_SIZE;
69
70         double* u;
71         double* v;
72         double* u_next;
73         double* v_next;
74
75         u = new double [vector_size];
76         v = new double [vector_size];

```

```

77     u_next = new double [vector_size];
78     v_next = new double [vector_size];
79
80     double delta_x=0.4; //space step
81     double delta_x_square = delta_x*delta_x; //squaring
to save computation later
82     double delta_t; //time step
83
84     //Define delta_t in relation to Dv to ensure
stability
85     if(Dv>1.09) delta_t =(delta_x*delta_x)/(6.01*Dv);
86     else delta_t=0.005;
87
88     double current_time = 0.0;
89     double final_time = 100;
90
91     int current_step=0;
92     int draw_interval=10; //Interval between drawing
steps
93     int offset = 0; //First timestep to draw
94
95     double activator, inhibitor, zero_level;
96
97     //Number of segments on each axis
98     int x_segments=20;
99     int y_segments=20;
100    double y_max = 1;
101    double y_min = -0.3;
102
103    double u_max=y_max;
104    double u_min=y_min;
105    double v_max=y_max;
106    double v_min=y_min;
107
108    //Interval between segments
109    double x_interval = MEDIUM_SIZE/x_segments;
110    double y_interval = (y_max-y_min)/y_segments;
111
112    //Index for for-loops
113    int i,j,ii,ip1,im1;
114

```

```

115     border = 0.05*min(R.Height(),R.Width());
116     top = R.TopLeft().y + border;
117     bottom = R.BottomRight().y - border;
118     plot_left = R.TopLeft().x + border;
119     plot_right = R.TopLeft().x + 0.5*R.Width() - border;
120     graph_right = R.BottomRight().x - border;
121     graph_left = R.TopLeft().x + 0.5*R.Width() + border;
122     plot_origin.x = int(floor((plot_right - plot_left -
MEDIUM_SIZE)*0.5));
123     plot_origin.y = int(floor((bottom - top -
MEDIUM_SIZE)*0.5));
124
125     x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
126     y_unit = (top-bottom)/(y_max-y_min);
127     graph_origin.x = graph_left;
128     graph_origin.y = bottom - y_min*y_unit;
129
130     //Du,Dv divided by delta_x^2 to save time
131     double Du_div = Du/(delta_x*delta_x*6);
132     double Dv_div = Dv/(delta_x*delta_x*6);
133
134     //Setting up initial conditions
135     for (i=0;i<MEDIUM_SIZE;i++)
136     {
137         ii=i*MEDIUM_SIZE;
138         for (j=0;j<MEDIUM_SIZE;j++)
139         {
140             u[ii + j]=0.0;
141             v[ii + j]=0.0;
142         }
143     }
144
145     for (i=(MEDIUM_SIZE/2 - 5);i<(MEDIUM_SIZE/2 + 5);i
++)
146     {
147         ii=i*MEDIUM_SIZE;
148         for (j=(MEDIUM_SIZE/2 - 5);j<(MEDIUM_SIZE/2 + 5)
;j++)
149         {
150             u[ii + j]=0.5;
151         }

```

```

152     }
153
154     while(current_time<final_time)
155     {
156         //Flux at boundaries
157         ii = MEDIUM_SIZE*MEDIUM_SIZE;
158         for(j=0;j<MEDIUM_SIZE;j++)
159         {
160             u[0 + j] = u[MEDIUM_SIZE + j];
161             u[ii-MEDIUM_SIZE + j] = u[ii-2*MEDIUM_SIZE +
j];
162             v[0 + j] = v[MEDIUM_SIZE + j];
163             v[ii-MEDIUM_SIZE + j] = v[ii-2*MEDIUM_SIZE +
j];
164         }
165         for(i=0;i<MEDIUM_SIZE;i++)
166         {
167             ii=i*MEDIUM_SIZE;
168             u[ii + 0] = u[ii + 1];
169             u[ii + MEDIUM_SIZE-1] = u[ii + MEDIUM_SIZE
-2];
170             v[ii + 0] = v[ii + 1];
171             v[ii + MEDIUM_SIZE-1] = v[ii + MEDIUM_SIZE
-2];
172         }
173
174         //Explicit Euler Method
175         for(i=1;i<MEDIUM_SIZE-1;i++)
176         {
177             ii=i*MEDIUM_SIZE;
178             ip1=(i+1)*MEDIUM_SIZE;
179             im1=(i-1)*MEDIUM_SIZE;
180             for(j=1;j<MEDIUM_SIZE-1;j++)
181             {
182                 u_next[ii + j] = u[ii + j] + delta_t*(
Du_div*(u[im1 + j-1] + 4*u[im1 + j] + u[im1 + j+1] + 4*u[
ii + j-1] -20*u[ii + j] +4*u[ii + j+1] + u[ip1 + j-1] +
4*u[ip1 + j] + u[ip1 + j+1]) - eu*(ku*u[ii + j]*(u[ii + j
]-u0)*(u[ii + j]-u1) + v[ii + j]));
183                 v_next[ii + j] = v[ii + j] + delta_t*(
Dv_div*(v[im1 + j-1] + 4*v[im1 + j] + v[im1 + j+1] + 4*v[

```



```

ii + j-1] -20*v[ii + j] +4*v[ii + j+1] + v[ip1 + j-1] +
4*v[ip1 + j] + v[ip1 + j+1]) + ev*(u[ii + j] - v[ii + j])
);
184         }
185     }
186
187     for (i=1;i<MEDIUM_SIZE-1;i++)
188     {
189         ii=i*MEDIUM_SIZE;
190         for (j=1;j<MEDIUM_SIZE-1;j++)
191         {
192             u[ii + j]=u_next[ii + j];
193             v[ii + j]=v_next[ii + j];
194         }
195     }
196
197     //Draw curves every "draw_interval"-steps
198     if(current_step%draw_interval==0)
199     {
200         y_interval = (y_max-y_min)/y_segments;
201         y_unit = (top-bottom)/(y_max-y_min);
202
203         //Clear plot area
204         dc.FillRect(plot_area,WHITE_BRUSH);
205
206         //Plot on left
207         //Draw bounding box
208         dc.SelectObject(blackPen);
209         dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
210         dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y-1);
211         dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y+MEDIUM_SIZE);
212         dc.LineTo(plot_origin.x-1,plot_origin.y+
MEDIUM_SIZE);
213         dc.LineTo(plot_origin.x-1,plot_origin.y-1);
214
215         for(i=0;i<MEDIUM_SIZE;i++)
216         {
217             ii=i*MEDIUM_SIZE;
218             for(j=0;j<MEDIUM_SIZE;j++)

```

```

219         {
220             activator = u[ii + j];
221             inhibitor = v[ii + j];
222
223             activator = int((activator-y_min)
*255/(y_max-y_min));
224             inhibitor = int((inhibitor-y_min)
*255/(y_max-y_min));
225             zero_level = int((0-y_min)*255/(
y_max-y_min));
226             pixel_colour = RGB(255-activator
,255-inhibitor,255-zero_level);
227             dc.SetPixel(plot_origin.x+j,
plot_origin.y+i, pixel_colour);
228         }
229     }
230
231     //Plot graph of centre cross section at
right
232     //Select blackPen
233     dc.SelectObject(blackPen);
234
235     //Draw x-axis
236     //centre row (this won't change)
237     ii = int(MEDIUM_SIZE*MEDIUM_SIZE*0.5);
238     if(y_min>=0)
239     {
240         dc.MoveTo(graph_left,bottom);
241         dc.LineTo(graph_right,bottom);
242     }
243     else
244     {
245         dc.MoveTo(graph_left,graph_origin.y);
246         dc.LineTo(graph_right,graph_origin.y);
247     }
248
249     //Draw y-axis
250     dc.MoveTo(graph_left,bottom);
251     dc.LineTo(graph_left,top);
252
253     //Label x-axis

```

```

254         if(y_min>=0)
255         {
256             s.x=graph_origin.x;
257             s.y=bottom;
258         }
259         else
260         {
261             s.x=graph_origin.x;
262             s.y=graph_origin.y;
263         }
264         for(j=0;j<=x_segments;j++)
265         {
266             r.y = s.y - 0.01*R.Height();
267             r.x = s.x + j*(graph_right-graph_left)/
x_segments;
268             dc.MoveTo(r.x,r.y);
269             r.y = s.y + 0.01*R.Height();
270             dc.LineTo(r.x,r.y);
271             str.Format(_T("%0.2f"),x_interval*j);
272             dc.TextOut(r.x-10,r.y+1,str);
273         }
274
275         //Label y-axis
276         for(j=0;j<=y_segments;j++)
277         {
278             r.x = graph_left - 0.01*R.Height();
279             r.y = bottom + j*(top-bottom)/y_segments
;
280             dc.MoveTo(r.x,r.y);
281             r.x = graph_left + 0.01*R.Height();
282             dc.LineTo(r.x,r.y);
283             str.Format(_T("%0.2f"),y_min +
y_interval*j);
284             dc.TextOut(r.x-50,r.y-7,str);
285         }
286
287         str.Format(_T("time=%0.2f"),current_time);
288         dc.TextOut(graph_left+70,top+30,str);
289         str.Format(_T("activator_ _blue"));
290         dc.TextOut(graph_left+70,top+60,str);
291         str.Format(_T("inhibitor_ _red"));

```

```

292         dc.TextOut(graph_left+70,top+75,str);
293
294         j=0;
295         dc.SelectObject(bluePen);
296         xj=(float)0;
297         r.x = graph_origin.x + xj*x_unit;
298         r.y = graph_origin.y + u[ii+j]*y_unit;
299         dc.MoveTo(r.x,r.y);
300
301         for (j=1;j<MEDIUM_SIZE;j++)
302         {
303             xj=(float)j;
304             r.x=graph_origin.x+xj*x_unit;
305             r.y=graph_origin.y+u[ii+j]*y_unit;
306             dc.LineTo(r.x,r.y);
307         }
308
309         j=0;
310         dc.SelectObject(redPen);
311         xj=(float)0;
312         r.x=graph_origin.x+xj*x_unit;
313         r.y=graph_origin.y+v[ii+j]*y_unit;
314         dc.MoveTo(r.x,r.y);
315
316         for (j=1;j<MEDIUM_SIZE;j++)
317         {
318             xj=(float)j;
319             r.x=graph_origin.x+xj*x_unit;
320             r.y=graph_origin.y+v[ii+j]*y_unit;
321             dc.LineTo(r.x,r.y);
322         }
323
324         olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
325         //Dump frame to tif file
326         if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
;
327     }
328

```

```

329         //Increment
330         current_time+=delta_t;
331         current_step++;
332     }
333 }
334 }
335
336 BOOL CReactionDiffusionDlg::DumpToFile(int width, int height
    , int istart, int jstart, CDC* dc, int count)
337 {
338     ...
339 }

```

Two-dimensional Fitzhugh-Nagumo Model with noise

```

1 // 2DFHN-with-noise.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "ReactionDiffusion.h"
6 #include "ReactionDiffusionDlg.h"
7 #include <math.h>
8 #include <afxwin.h>
9 #include <stdlib.h>
10
11 #ifdef _DEBUG
12 #define new DEBUG_NEW
13 #endif
14
15 void CReactionDiffusionDlg::OnPaint()
16 {
17     CPaintDC olddc(this); // device context for painting
18     CDC dc; //We will bitblit this to olddc
19     CBitmap bmpdc;
20
21     if (IsIconic())
22     {
23         ...
24     }
25     else
26     {

```

```

27         CDialog::OnPaint();
28
29         //Get plot area and save dimensions
30         CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
31         CRect R; pWnd->GetWindowRect(&R);
32         INT nPlotWidth = R.Width();
33         INT nPlotHeight = R.Height();
34
35         ScreenToClient(&R);
36         CPoint r,s,plot_origin,graph_origin;
37         LPCRECT plot_area(R);
38
39         COLORREF pixel_colour = RGB(0,0,0);
40
41         dc.CreateCompatibleDC(&dc);
42         bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
Height());
43         dc.SelectObject(&bmpdc);
44
45         double plot_left,plot_right,graph_left,graph_right,
top,bottom,x_unit,y_unit,border,xj;
46         CString str;
47
48         const int MEDIUM_SIZE = 200; //Size of medium:
MEDIUM_SIZE * MEDIUM_SIZE
49
50         bool dump = false; //Dump each frame to a tif file?
51
52         const double Du = 1.0; //Diffusion coefficient of u
53         double Dv = 2.0; //Diffusion coefficient of v
54         double eu = 1.0; //Rate of kinetics of u
55         double ev = 0.2; //Rate of kinetics of v
56         double ku = 4.5;
57         double u1 = 1;
58         double u0 = 0.05;
59
60         double stim,a,init;
61
62         //Seed for RNG
63         int seed = 123;
64

```

```

65         double ju0 = 0.0; //boundary flux of u at i=0
66         double ju1 = 0.0; //boundary flux of u at i=
MEDIUM_SIZE-1
67
68         double jv0 = 0.0; //boundary flux of v at i=0
69         double jv1 = 0.0; //boundary flux of v at i=
MEDIUM_SIZE-1
70
71         int vector_size = MEDIUM_SIZE*MEDIUM_SIZE;
72
73         double* u;
74         double* v;
75         double* u_next;
76         double* v_next;
77
78         u = new double [vector_size];
79         v = new double [vector_size];
80         u_next = new double [vector_size];
81         v_next = new double [vector_size];
82
83         double delta_x=0.4; //space step
84         double delta_x_square = delta_x*delta_x; //squaring
to save computation later
85         double delta_t; //time step
86
87         //Define delta_t in relation to Dv to ensure
stability
88         if(Dv>1.09) delta_t =(delta_x*delta_x)/(6.01*Dv);
89         else delta_t=0.005;
90
91         double current_time = 0.0;
92         double final_time = 100;
93
94         int current_step=0;
95         int draw_interval=50; //Interval between drawing
steps
96         int offset = 0; //First timestep to draw
97
98         double activator, inhibitor, zero_level;
99
100        //Number of segments on each axis

```

```

101     int x_segments=20;
102     int y_segments=20;
103     double y_max = 1;
104     double y_min = -0.3;
105
106     double u_max=y_max;
107     double u_min=y_min;
108     double v_max=y_max;
109     double v_min=y_min;
110
111     //Interval between segments
112     double x_interval = MEDIUM_SIZE/x_segments;
113     double y_interval = (y_max-y_min)/y_segments;
114
115     //Index for for-loops
116     int i,j,ii,ip1,im1;
117
118     border = 0.05*min(R.Height(),R.Width());
119     top = R.TopLeft().y + border;
120     bottom = R.BottomRight().y - border;
121     plot_left = R.TopLeft().x + border;
122     plot_right = R.TopLeft().x + 0.5*R.Width() - border;
123     graph_right = R.BottomRight().x - border;
124     graph_left = R.TopLeft().x + 0.5*R.Width() + border;
125     plot_origin.x = int(floor((plot_right - plot_left -
MEDIUM_SIZE)*0.5));
126     plot_origin.y = int(floor((bottom - top -
MEDIUM_SIZE)*0.5));
127
128     x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
129     y_unit = (top-bottom)/(y_max-y_min);
130     graph_origin.x = graph_left;
131     graph_origin.y = bottom - y_min*y_unit;
132
133     //Du,Dv divided by delta_x^2 to save time
134     double Du_div = Du/(delta_x*delta_x*6);
135     double Dv_div = Dv/(delta_x*delta_x*6);
136
137     //Setting up initial conditions
138     for (i=0;i<MEDIUM_SIZE;i++)
139     {

```



```

140         ii=i*MEDIUM_SIZE;
141         for (j=0;j<MEDIUM_SIZE;j++)
142         {
143             u[ii + j]=0.0;
144             v[ii + j]=0.0;
145         }
146     }
147
148     srand(seed);
149     a=0;
150
151     while(current_time<final_time)
152     {
153         //Apply noise every 100 time steps
154         if(current_step%500==0){
155             init = (double) rand();
156             init = init/RAND_MAX;
157             init = init*vector_size;
158
159             init_coord.x = ((int) floor(init))%
MEDIUM_SIZE;
160             init_coord.y = floor(init/MEDIUM_SIZE);
161
162             a = (double) rand();
163             a = a/RAND_MAX;
164             a = a - 0.5;
165         }
166
167         //Flux at boundaries
168         ii = MEDIUM_SIZE*MEDIUM_SIZE;
169         for(j=0;j<MEDIUM_SIZE;j++)
170         {
171             u[0 + j] = u[MEDIUM_SIZE + j];
172             u[ii-MEDIUM_SIZE + j] = u[ii-2*MEDIUM_SIZE +
j];
173             v[0 + j] = v[MEDIUM_SIZE + j];
174             v[ii-MEDIUM_SIZE + j] = v[ii-2*MEDIUM_SIZE +
j];
175         }
176         for(i=0;i<MEDIUM_SIZE;i++)
177         {

```

```

178         ii=i*MEDIUM_SIZE;
179         u[ii + 0] = u[ii + 1];
180         u[ii + MEDIUM_SIZE-1] = u[ii + MEDIUM_SIZE
-2];
181         v[ii + 0] = v[ii + 1];
182         v[ii + MEDIUM_SIZE-1] = v[ii + MEDIUM_SIZE
-2];
183     }
184
185     //Explicit Euler Method
186     for(i=1;i<MEDIUM_SIZE-1;i++)
187     {
188         ii=i*MEDIUM_SIZE;
189         ip1=(i+1)*MEDIUM_SIZE;
190         im1=(i-1)*MEDIUM_SIZE;
191         for(j=1;j<MEDIUM_SIZE-1;j++)
192         {
193             if(current_step%500<100 && ((init_coord.
x - i)*(init_coord.x - i) + (init_coord.y - j)*(
init_coord.y - j))<25){
194                 stim = a*0.4;
195             }else{
196                 stim = 0;
197             }
198
199             u_next[ii + j] = u[ii + j] + delta_t*(
Du_div*(u[im1 + j-1] + 4*u[im1 + j] + u[im1 + j+1] + 4*u[
ii + j-1] -20*u[ii + j] +4*u[ii + j+1] + u[ip1 + j-1] +
4*u[ip1 + j] + u[ip1 + j+1]) - eu*(ku*u[ii + j]*(u[ii + j
]-u0)*(u[ii + j]-u1) + v[ii + j])) + stim);
200             v_next[ii + j] = v[ii + j] + delta_t*(
Dv_div*(v[im1 + j-1] + 4*v[im1 + j] + v[im1 + j+1] + 4*v[
ii + j-1] -20*v[ii + j] +4*v[ii + j+1] + v[ip1 + j-1] +
4*v[ip1 + j] + v[ip1 + j+1]) + ev*(u[ii + j] - v[ii + j])
);
201         }
202     }
203
204     for (i=1;i<MEDIUM_SIZE-1;i++)
205     {
206         ii=i*MEDIUM_SIZE;

```

```

207         for (j=1;j<MEDIUM_SIZE-1;j++)
208         {
209             u[ii + j]=u_next[ii + j];
210             v[ii + j]=v_next[ii + j];
211         }
212     }
213
214     //Draw curves every "draw_interval"-steps
215     if(current_step%draw_interval==0)
216     {
217         y_interval = (y_max-y_min)/y_segments;
218         y_unit = (top-bottom)/(y_max-y_min);
219
220         //Clear plot area
221         dc.FillRect(plot_area,WHITE_BRUSH);
222
223         //Plot on left
224         //Draw bounding box
225         dc.SelectObject(blackPen);
226         dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
227         dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y-1);
228         dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y+MEDIUM_SIZE);
229         dc.LineTo(plot_origin.x-1,plot_origin.y+
MEDIUM_SIZE);
230         dc.LineTo(plot_origin.x-1,plot_origin.y-1);
231
232         for(i=0;i<MEDIUM_SIZE;i++)
233         {
234             ii=i*MEDIUM_SIZE;
235             for(j=0;j<MEDIUM_SIZE;j++)
236             {
237                 activator = u[ii + j];
238                 inhibitor = v[ii + j];
239
240                 activator = int((activator-y_min)
*255/(y_max-y_min));
241                 inhibitor = int((inhibitor-y_min)
*255/(y_max-y_min));

```

```

242         zero_level = int((0-y_min)*255/(
y_max-y_min));
243         pixel_colour = RGB(255-activator
,255-inhibitor,255-zero_level);
244         dc.SetPixel(plot_origin.x+j,
plot_origin.y+i, pixel_colour);
245     }
246 }
247
248     //Plot graph of centre cross section at
right
249     //Select blackPen
250     dc.SelectObject(blackPen);
251
252     //Draw x-axis
253     //centre row (this won't change)
254     ii = int(MEDIUM_SIZE*MEDIUM_SIZE*0.5);
255     if(y_min>=0)
256     {
257         dc.MoveTo(graph_left,bottom);
258         dc.LineTo(graph_right,bottom);
259     }
260     else
261     {
262         dc.MoveTo(graph_left,graph_origin.y);
263         dc.LineTo(graph_right,graph_origin.y);
264     }
265
266     //Draw y-axis
267     dc.MoveTo(graph_left,bottom);
268     dc.LineTo(graph_left,top);
269
270     //Label x-axis
271     if(y_min>=0)
272     {
273         s.x=graph_origin.x;
274         s.y=bottom;
275     }
276     else
277     {
278         s.x=graph_origin.x;

```

```

279         s.y=graph_origin.y;
280     }
281     for(j=0;j<=x_segments;j++)
282     {
283         r.y = s.y - 0.01*R.Height();
284         r.x = s.x + j*(graph_right-graph_left)/
x_segments;
285         dc.MoveTo(r.x,r.y);
286         r.y = s.y + 0.01*R.Height();
287         dc.LineTo(r.x,r.y);
288         str.Format(_T("%0.2f"),x_interval*j);
289         dc.TextOut(r.x-10,r.y+1,str);
290     }
291
292     //Label y-axis
293     for(j=0;j<=y_segments;j++)
294     {
295         r.x = graph_left - 0.01*R.Height();
296         r.y = bottom + j*(top-bottom)/y_segments
;
297         dc.MoveTo(r.x,r.y);
298         r.x = graph_left + 0.01*R.Height();
299         dc.LineTo(r.x,r.y);
300         str.Format(_T("%0.2f"),y_min +
y_interval*j);
301         dc.TextOut(r.x-50,r.y-7,str);
302     }
303
304     str.Format(_T("time=%0.2f"),current_time);
305     dc.TextOut(graph_left+70,top+30,str);
306     str.Format(_T("activator_□_□blue"));
307     dc.TextOut(graph_left+70,top+60,str);
308     str.Format(_T("inhibitor_□_□red"));
309     dc.TextOut(graph_left+70,top+75,str);
310
311     j=0;
312     dc.SelectObject(bluePen);
313     xj=(float)0;
314     r.x = graph_origin.x + xj*x_unit;
315     r.y = graph_origin.y + u[ii+j]*y_unit;
316     dc.MoveTo(r.x,r.y);

```

```

317
318         for (j=1;j<MEDIUM_SIZE;j++)
319         {
320             xj=(float)j;
321             r.x=graph_origin.x+xj*x_unit;
322             r.y=graph_origin.y+u[ii+j]*y_unit;
323             dc.LineTo(r.x,r.y);
324         }
325
326         j=0;
327         dc.SelectObject(redPen);
328         xj=(float)0;
329         r.x=graph_origin.x+xj*x_unit;
330         r.y=graph_origin.y+v[ii+j]*y_unit;
331         dc.MoveTo(r.x,r.y);
332
333         for (j=1;j<MEDIUM_SIZE;j++)
334         {
335             xj=(float)j;
336             r.x=graph_origin.x+xj*x_unit;
337             r.y=graph_origin.y+v[ii+j]*y_unit;
338             dc.LineTo(r.x,r.y);
339         }
340
341         olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
342         //Dump frame to tif file
343         if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
;
344     }
345
346     //Increment
347     current_time+=delta_t;
348     current_step++;
349 }
350 }
351 }
352

```

```
353 | BOOL CReactionDiffusionDlg::DumpToFile(int width, int height  
    | , int istart, int jstart, CDC* dc, int count)  
354 | {  
355 | ...  
356 | }
```