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

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.

$$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_2} E + P \tag{2.1}$$

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:

$$\frac{ds}{dt} = -k_1 es + k_{-1}c \tag{2.2}$$

$$\frac{dc}{dt} = k_1 es - (k_{-1} + k_2)c \tag{2.3}$$

$$\frac{de}{dt} = -k_1 es + (k_{-1} + k_2)c \tag{2.4}$$

$$\frac{dp}{dt} = k_2 c \tag{2.5}$$

Adding (2.3) and (2.4) we get:

$$\frac{de}{dt} + \frac{dc}{dt} = 0 \tag{2.6}$$

$$\Rightarrow e + c = e_0 \tag{2.7}$$

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):

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

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

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

We non-dimensionalise with:

$$\tau = tk_1 e_0 \tag{2.10}$$

$$u = \frac{s}{s_0} \tag{2.11}$$

$$v = \frac{c}{e_0} \tag{2.12}$$

$$\epsilon = \frac{e_0}{s_0} \tag{2.13}$$

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

To obtain:

$$\frac{ds}{d\tau}\frac{d\tau}{dt} = k_1 e_0 \frac{ds}{d\tau} = -k_1 e_0 s + (k_1 s + k_{-1})c \tag{2.14}$$

$$k_1 e_0 \frac{dc}{d\tau} = k_1 e_0 s - (k_1 s + k_{-1} + k_2)c \tag{2.15}$$

Then with $ds = s_0 du$, $dc = e_0 dv$

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

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

Dividing both by $k_1 e_0 s_0$:

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

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

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:

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

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

From here there are two approaches for analysis.

2.1.2 Phase-Plane Analysis

Null-clines for the u-v plane are:

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

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

We consider different behaviours in two time periods; the intial very short time period where $0 < \tau << 1$ 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 ${}^{dv}\!/_{dt} = 0$.

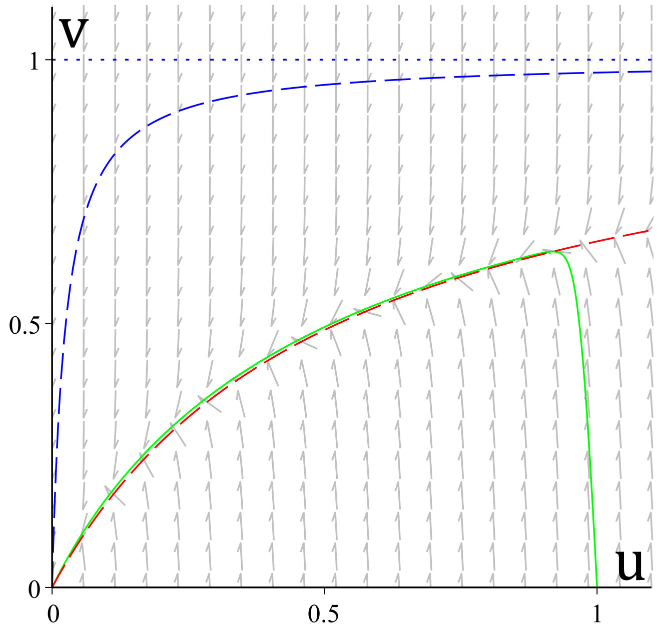

Figure 2.1: Phase Portrait for System 2.20, 2.21
Green line shows a phase trajectory starting at (1,0). The dashed blue line is the null cline $^{du}/_{dt} = 0$. *The dashed red line is the null cline* $^{dv}/_{dt} = 0$. *Horizontal dotted blue line is the asymptote* $v = 1$. *This particular example 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.*

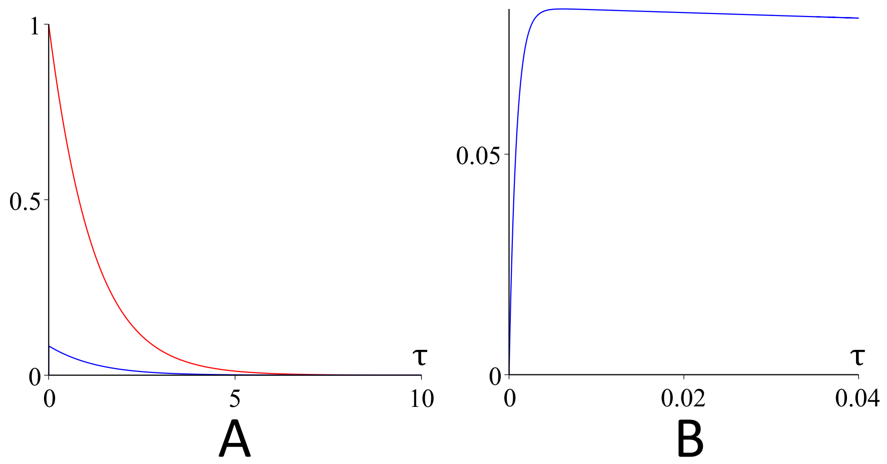

Figure 2.2: Dynamics of variables u and v during enzymatic reaction
A: *u (red) and v (blue) plotted against time.*
B: *detail of **A** for small time values. Notice the rapid increase in the concentration of v over the short initial time period.*

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

$$\frac{dv}{d\tau} = \frac{1}{\epsilon}(1 - (1+K)v) \tag{2.24}$$
$$= \frac{1}{\epsilon} - \frac{1+K}{\epsilon}v \tag{2.25}$$

Suppose that

$$v(\tau) = A + Be^{-\frac{1+K}{\epsilon}\tau} \tag{2.26}$$
$$\Rightarrow v(\tau) = \frac{1}{1+K}(1 - e^{-\frac{1+K}{\epsilon}\tau}) \tag{2.27}$$

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 $v = \frac{u}{u+K} = 1 - \frac{K}{u+K}$:

$$\frac{du}{d\tau} = -u + (u + K - \lambda)(1 - \frac{K}{u+K}) \tag{2.28}$$
$$= -u + u + K - \lambda - \frac{(u+K-\lambda)K}{u+K} \tag{2.29}$$
$$= K - \lambda - K + \frac{\lambda K}{u+K} \tag{2.30}$$
$$= -\frac{\lambda u}{u+K} \tag{2.31}$$
$$\Rightarrow \frac{u+K}{u}du = -\lambda d\tau_r \tag{2.32}$$
$$\Rightarrow u + Klnu = -\lambda\tau_r + 1 \tag{2.33}$$

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

$$\frac{1}{e} - K = -\lambda\tau_{relu} + 1 \tag{2.34}$$
$$\Rightarrow \tau_{relu} = \frac{1 - {}^1\!/_e + K}{\lambda} \tag{2.35}$$

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

$$\frac{\tau_{relv}}{\tau_{relu}} = \frac{\epsilon\lambda}{(1+K)(1 - {}^1\!/_e + K)} \propto \epsilon \ll 1 \tag{2.36}$$

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.

$$\frac{dv}{d\tau} = 0 \tag{2.37}$$

$$\Rightarrow u - (u + K)v = 0 \tag{2.38}$$

$$\Rightarrow v = \frac{u}{u + K} \tag{2.39}$$

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

$$\frac{dp}{dt} = k_2e_0\frac{u}{u + K} \tag{2.40}$$

$$\Rightarrow \frac{dp}{dt} = k_2e_0\frac{s}{s + Ks_0} \tag{2.41}$$

$$\Rightarrow \frac{dp}{dt} = R = \frac{Qs}{s + K_m} \tag{2.42}$$

Where $Q = k_2e_0$ 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.

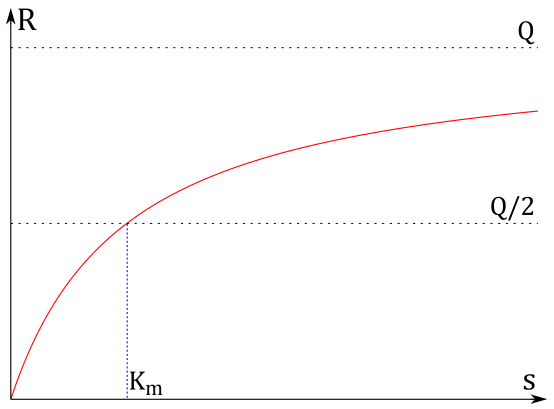

Figure 2.3: Reaction rate as a function of concentration of the substrate

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

$$\frac{Q}{2} = \frac{k_2 e_0}{2} = k_2 e_0 \frac{s}{s + K_m} \tag{2.43}$$

$$\Rightarrow \frac{1}{2} = \frac{s}{s + K_m} \tag{2.44}$$

$$\Rightarrow s + K_m = 2s \tag{2.45}$$

$$\Rightarrow s = K_m \tag{2.46}$$

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.

$$2S + E \rightleftharpoons C \rightarrow E + 2P \tag{2.47}$$

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.

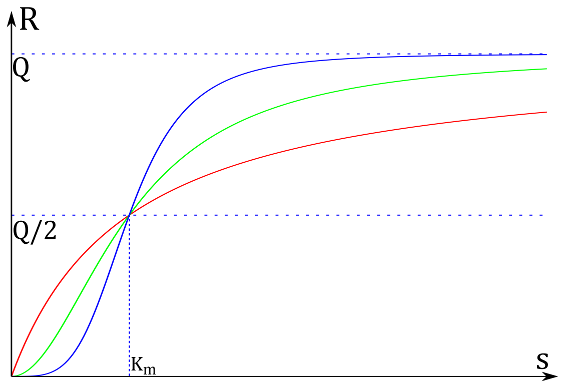

Figure 2.4: Reaction rate as a function of concentration of the substrate
Red - $R(s)$ for the simple case. 2.42
Green - $R(s)$ for cooperative case (requiring two molecules). 2.47
Blue - $R(s)$ for cooperative case (requiring four molecules).
For these cooperative cases, a sigmoidal shape to the graph 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.

$$S + E \rightleftharpoons C_1 \rightarrow E + P \tag{2.48}$$

$$S + C_1 \rightleftharpoons C_2 \rightarrow C_1 + P \tag{2.49}$$

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:

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

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

Again, this corresponds to a point-system or a solution which is homogenised. If we define $f(u, v) = -\epsilon_u(k_u u(u - u_o)(u - u_1) + v)$ 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.

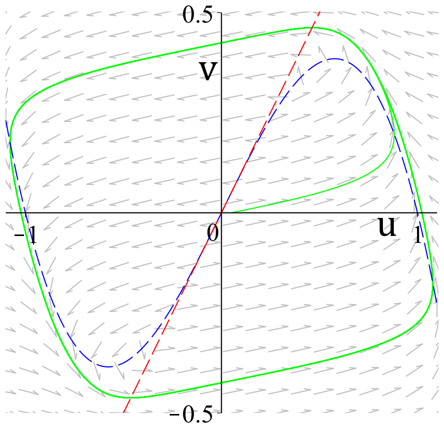

Figure 2.5: Oscillatory behaviour in the FitzHugh-Nagumo system
Here for demonstrative purposes $u_0 = -1$, $u_1 = 1$, $k_u = 4.5$. The blue dashed line shows the null cline ${}^{du}/_{dt} = 0$, the red dashed line the null cline ${}^{dv}/_{dt} = 0$. The green line is an example trajectory starting at $(0.05, 0)$, a small perturbation from the single unstable equilibrium at $(0, 0)$. The orbit which it falls onto is the limit cycle for this system.

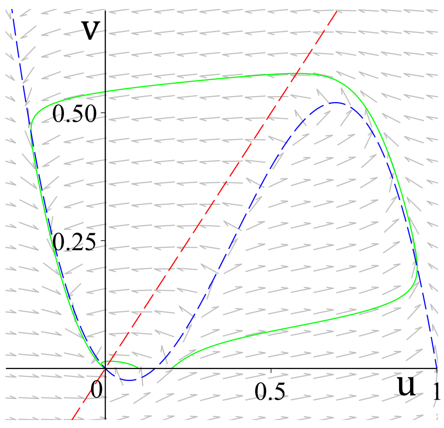

Figure 2.6: Excitable behaviour in the FitzHugh-Nagumo system
Here we have $u_0 = 0.15$, $u_1 = 1$, $k_u = 5$. There is only one equilibrium which is stable. Again, the blue dashed line shows the null cline ${}^{du}/_{dt} = 0$, the red dashed line the null cline ${}^{dv}/_{dt} = 0$. The green lines here 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.

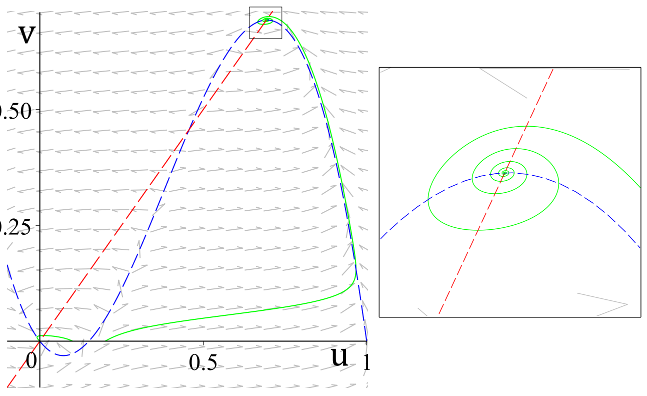

Figure 2.7: Multistable behaviour in the FitzHugh-Nagumo system
Here we have $u_0 = 0.15$, $u_1 = 1$, $k_u = 6$. *There are now three equilibria of which two are stable, one is unstable. Again, the blue dashed line shows the null cline* $^{du}/_{dt} = 0$, *the red dashed line the null cline* $^{dv}/_{dt} = 0$. *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 the 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.

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + f(u, v) \tag{2.52}$$
$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + g(u, v) \tag{2.53}$$

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:

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

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

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

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

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

With constants C_1 and C_2 defined by the boundary conditions.
For convenience, define $\lambda = \sqrt{D/k}$:

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

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:

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

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

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

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:

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

Again, substituting *relative position* ξ:

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

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:

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

Again, substituting *relative position* ξ:

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

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

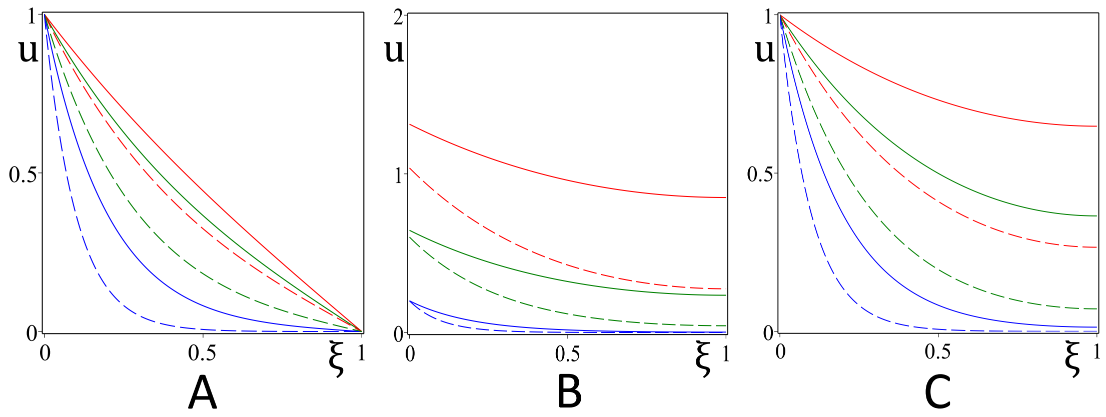

Figure 3.1: Profiles of u for different boundary conditions
In each of ***A***, ***B***, ***C***: $L = 1, 2$ *- solid, dashed lines respectively.* $\lambda = 1, 0.6, 0.2$ *- red, green, blue lines respectively.*
A: *profile of* u *for Dirichlet BCs.*
B: *profile of* u *for Neumann BCs.*
C: *profile of* 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 condtions, 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 decribed 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).

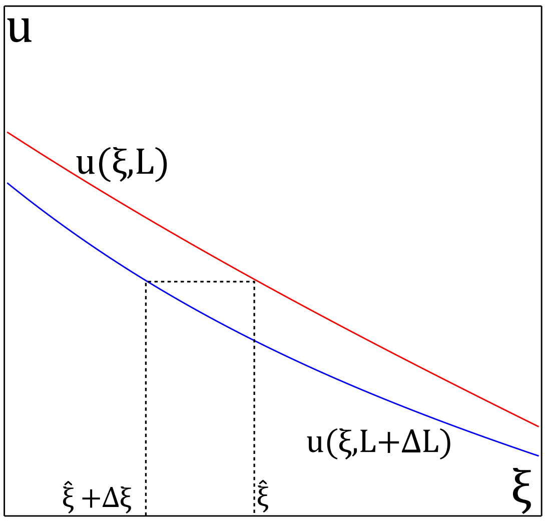

Figure 3.2: Derivation of the Scaling Factor
Two non-identical profiles of u for differing lengths of of the medium, L and $L + \Delta L$. The concentrations are equal at $u(\hat{\xi}, L) = u(\hat{\xi} + \Delta\xi, L + \Delta L)$

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

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

$$\frac{\partial u}{\partial \xi}\Delta\xi + \frac{\partial u}{\partial L}\Delta L = 0 \tag{3.12}$$

$$\Rightarrow \Delta\xi = -\frac{\frac{\partial u}{\partial L}}{\frac{\partial u}{\partial \xi}}\Delta L \tag{3.13}$$

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

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

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:

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

For Neumann Boundary Conditions:

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

For Mixed Boundary Conditions:

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

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 is S is generally negative, for simplicity we plot the absolute value $|S|$.

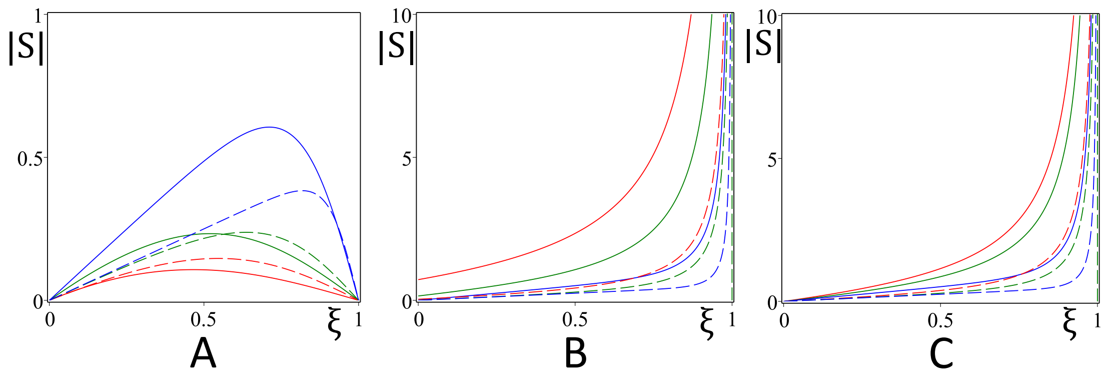

Figure 3.3: Scaling factor, $|S|$ as a function of relative position for three sets of boundary conditions
A: $\left|S(\xi)\right|$ *for Dirichlet BCs.*
B: $\left|S(\xi)\right|$ *for Neumann BCs.*
C: $\left|S(\xi)\right|$ *for mixed BCs.*
*In each of **A**, **B**, **C**:*
$L = 1, 2$ *- solid, dashed 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.

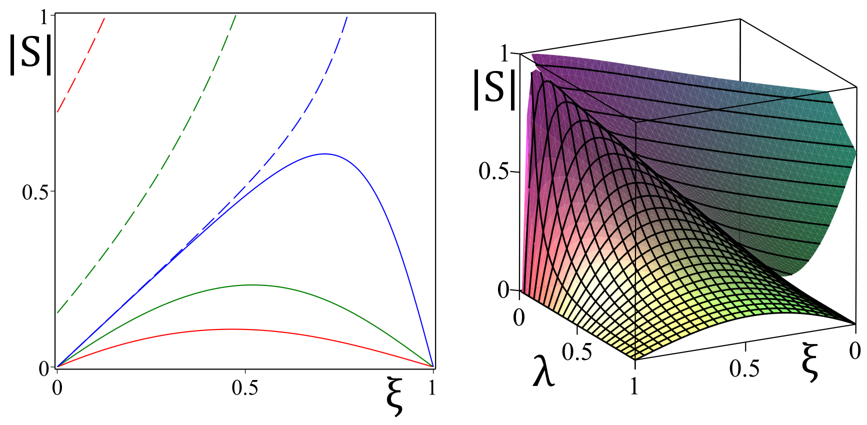

Figure 3.4: Scaling Factor for Dirichlet and Neumann boundary conditions for $L = 1$ ***Left:****Dirichlet Boundary Conditions - solid line*
Neumann Boundary Conditions - dashed line
$\lambda = 1, 0.6, 0.2$ *(red, green and blue lines respectively)*
This also serves as a cross-section of the plot at 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:

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

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 improves 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 qualititively 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

$$\lim_{\lambda \to 0} \chi = 1 \tag{3.19a}$$

$$\lim_{\lambda \to \sim 0.75} \chi = 0 \tag{3.19b}$$

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

$$\lim_{L \to 1} \chi = 0 \tag{3.20a}$$

$$\lim_{L \to \infty} \chi = 1 \tag{3.20b}$$

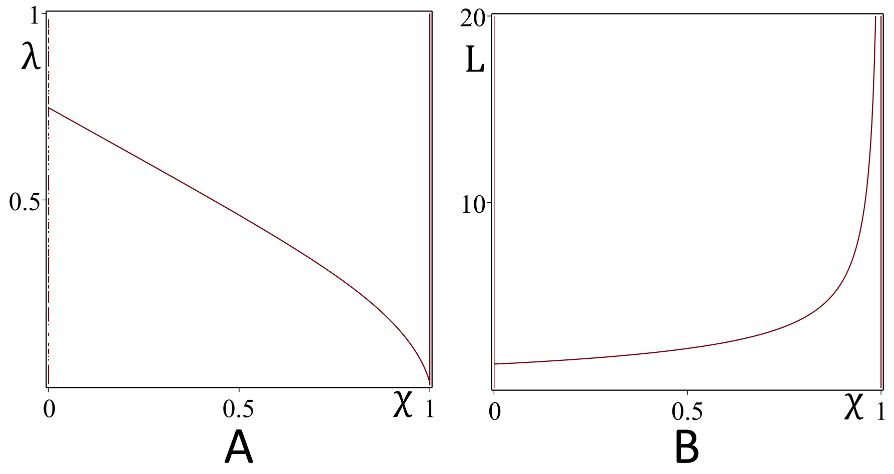

Figure 3.5: Behaviour of χ, the point at which scaling is not changing for Dirichlet boundary conditions.
A:*χ for constant L (here, $L = 1$) with $\lambda \in (0, 1)$.*
B:*χ for constant λ (here, $\lambda = 1$) with $L \in (0, 20)$.*
The formulae for $\frac{\partial S}{\partial L}$ were calculated using 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

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

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

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

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

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

with constants ρ, μ_u, μ_v, ρ_u, ρ_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:

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

$$g(u,v) = \epsilon_v(u-v) \tag{4.6}$$

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.

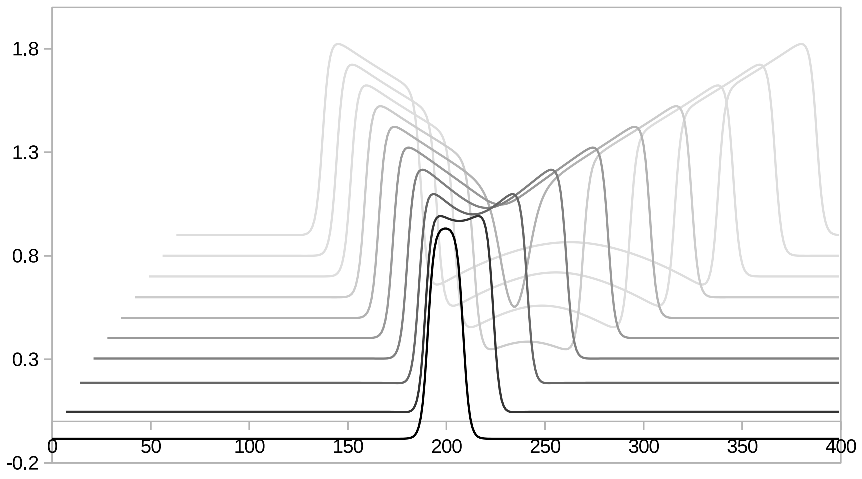

Figure 4.1: Travelling Wave in the FHN model
Only the concentration of the activator is shown here for clarity. The initial spot splits into a pair of propagating waves. In this figure, profiles at later points in time are shown in decreasingly paler shades of grey, recessed in a pseudo-z-axis.

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.

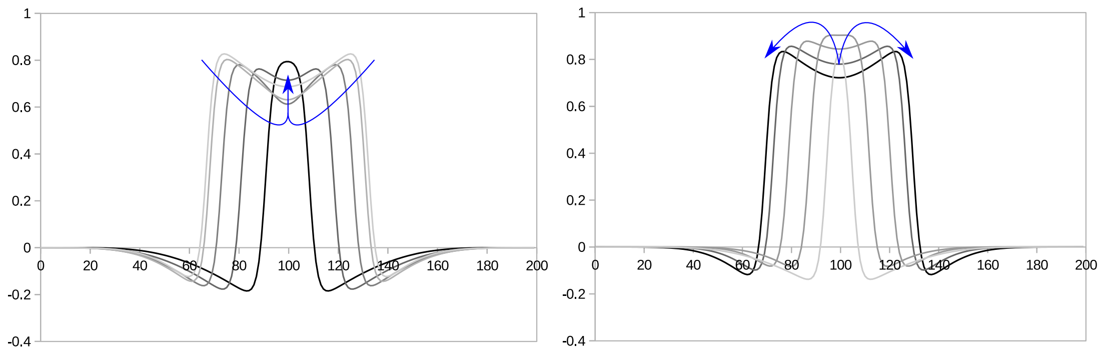

Figure 4.2: Pulsating Spot in the FHN model
Only the concentration of the activator is shown here for clarity. Notice that the spot oscillates in amplitude as well as width. The profiles for later points in time are in darker shades of grey.

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-hommogeneous equilibrium, a stationary spot. Again the postion of the spot does not change, but the oscillations dampen down to this stable equilibrium. Larger values of D_v give wider spots.

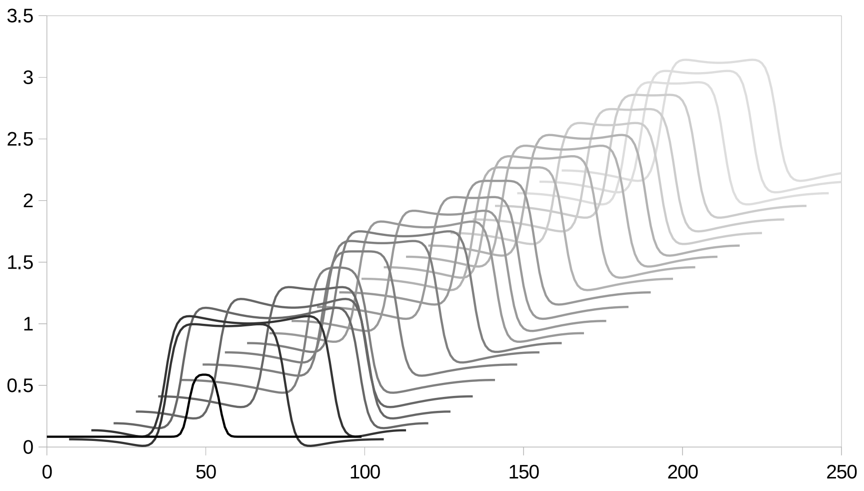

Figure 4.3: Stationary spots in the FitzHugh-Nagumo System
$D_v = 4$ *- The spot oscillates to a stable equililbrium. Profiles for greater values of time are shown in paler shades of grey.*

With changing *epsilon*$_v$, 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.

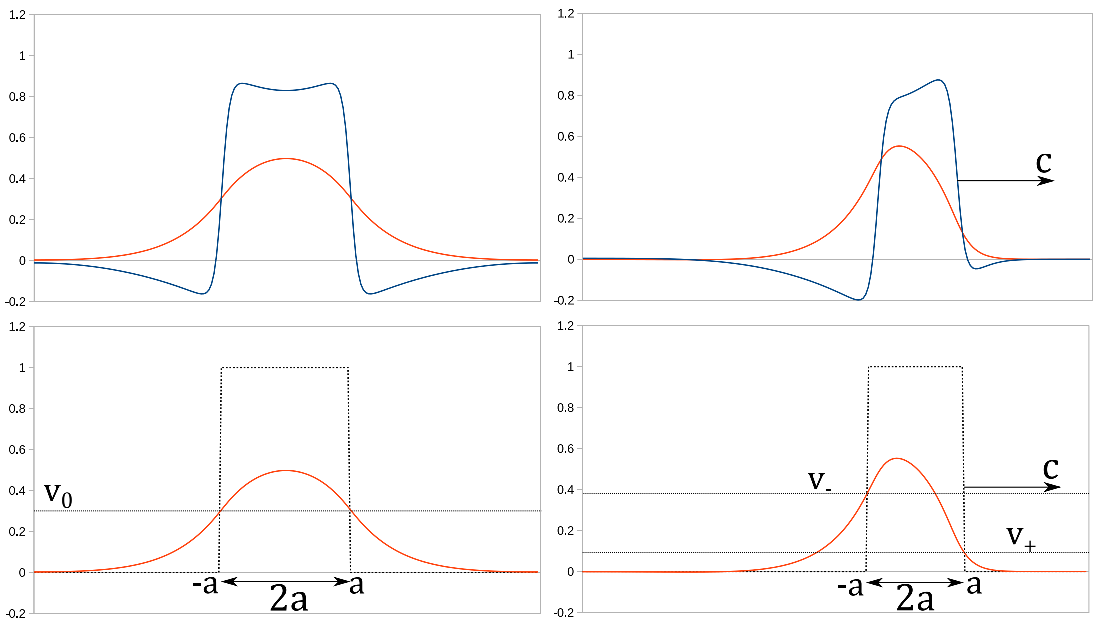

Figure 4.4: Approximation of the stationary spot and travelling wave
Replacing the profile for u with the rectangular function $u = 1$ for $|x| \leq a$, and $u = 0$ for $|x| > a$ where $2a$ is the width of the spot.

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

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

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

Our boundary conditions are as follows:

$$v(\infty) = v(-\infty) = 0 \tag{4.9}$$
$$v(a) = v(-a) = v_0 \tag{4.10}$$

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

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

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:

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

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

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

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

Where f and g are functions to be determined.

Then introducing a perturbation:

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

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

Which gives the first-order linearisation:

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

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

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

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

$$= b_{11}\delta a + b_{12}\delta v \tag{4.20}$$

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

$$= b_{21}\delta a + b_{22}\delta v \tag{4.22}$$

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

$$\begin{pmatrix} \delta a \\ \delta v \end{pmatrix}' = B \begin{pmatrix} \delta a \\ \delta v \end{pmatrix} \tag{4.23}$$

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

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

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).

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

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

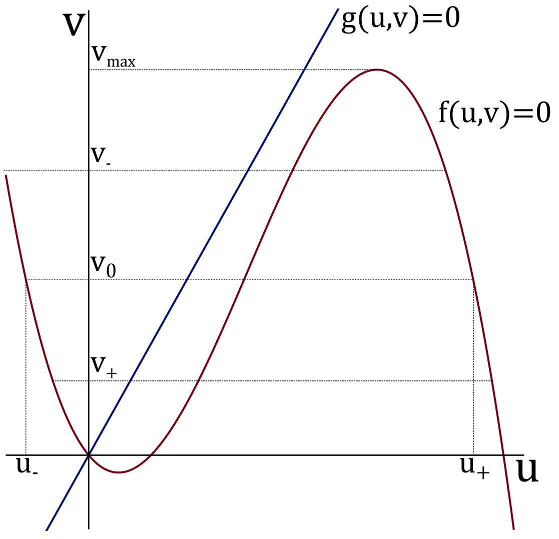

Figure 4.5: Nullclines $f(u, v) = 0$, $g(u, v) = 0$ for the Fitzhugh-Nagumo system

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

we have:

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

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:

$$\begin{aligned} c(v_0 + \delta v) &\approx c(v_0) + c'(v_0)\delta v & (4.27) \\ &= c'(v_0)\delta v & (4.28) \end{aligned}$$

Then for δv

$$\delta v = v(a + \delta a) - v(a) \tag{4.29}$$

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

$$\delta v \approx [v(a) + v'(a)\delta a] - v(a) \qquad = v'(a)\delta a \tag{4.30}$$

So we have:

$$\begin{aligned} c(v_0 + \delta v) &\approx c'(v_0)v'(a)\delta a & (4.31) \\ \Rightarrow b_{11} &= c'(v_0)v'(a) & (4.32) \end{aligned}$$

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

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

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

$$\begin{aligned} \frac{d}{dt}(v_0 + \delta v) &= D_v \Delta(v_0 + \delta v) + \epsilon_v(u_0 - [v_0 + \delta v]) & (4.34) \\ &= D_v \Delta v_0 + \epsilon_v(u_0 - v_0) + D_v \Delta \delta v - \epsilon_v \delta v & (4.35) \\ &= \underbrace{\frac{dv_0}{dt}}_{=0} + \frac{d\delta v}{dt} & (4.36) \end{aligned}$$

Where Δ is the Laplacian operator.

The spatial distribution can be represented as a fourier series,

$$\delta v = \sum_{k=0}^{\inf} e^{\gamma(k)t} coskx \tag{4.37}$$

$$\begin{aligned}
\Rightarrow \frac{d\delta v}{dt} &= \frac{d}{dt} e^{\gamma(k)t} cos(kx) = D_v \Delta e^{\gamma(k)t} coskx - \epsilon_v e^{\gamma(k)t} coskx && (4.38)\\
&= -D_v k^2 e^{\gamma(k)t} coskx - \epsilon_v e^{\gamma(k)t} coskx && (4.39)\\
&= -(D_v k^2 + \epsilon_v) e^{\gamma(k)t} coskx && (4.40)\\
&= -(D_v k^2 + \epsilon_v)\delta v && (4.41)
\end{aligned}$$

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

$$\begin{aligned}
tr(B) &= b_{11} + b_{22} && (4.42)\\
&= -\lambda v_0 c'(v_0) - (D_v k^2 + \epsilon_v) && (4.43)
\end{aligned}$$

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

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

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

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

for stability.

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

For b_{12}:

$$\begin{aligned}
\frac{da}{dt} &= c(v) \approx c(v_0 + \delta v) && (4.46)\\
&= \underbrace{c(v_0)}_{=0} + c'(v_0)\delta v && (4.47)
\end{aligned}$$

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

Then for b_{21}:

$$b_{21}\delta a = \epsilon_v(v_0(a+\delta a) - v(a+\delta a)) \tag{4.48}$$
$$= \epsilon_v((v_0 + (1+2v_0)\lambda\delta a) - v_0(1-\lambda\delta a)) \tag{4.49}$$
$$= \epsilon_v(\lambda\delta a - 2v_0\lambda\delta a + v_0\lambda\delta a) \tag{4.50}$$
$$= \epsilon_v(\lambda\delta a - v_0\lambda\delta a) \tag{4.51}$$
$$= \epsilon_v\lambda(1-v_0)\delta a \tag{4.52}$$
$$\Rightarrow b_{21} = \epsilon_v\lambda(1-v_0) \tag{4.53}$$

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

$$\lambda v_0 c'(v_0)(D_v k^2 + \epsilon_v) - c'(v_0)\epsilon_v(1-v_0)\lambda > 0 \tag{4.54}$$
$$\tag{4.55}$$

Then with mode $k = 0$ again:

$$\Rightarrow \epsilon_v\lambda v_0 c'(v_0) - c'(v_0)\epsilon_v\lambda + c'(v_0)\epsilon_v v_0\lambda > 0 \tag{4.56}$$

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

$$v_0 < 0.5 \tag{4.57}$$

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

So our conditions for stability are:

$$D_v < \frac{v_0^2}{\epsilon_v}c'(v_0)^2 \propto \frac{1}{\epsilon_v} \tag{4.58}$$
$$v_0 < 0.5 \tag{4.59}$$

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 sufficent 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.

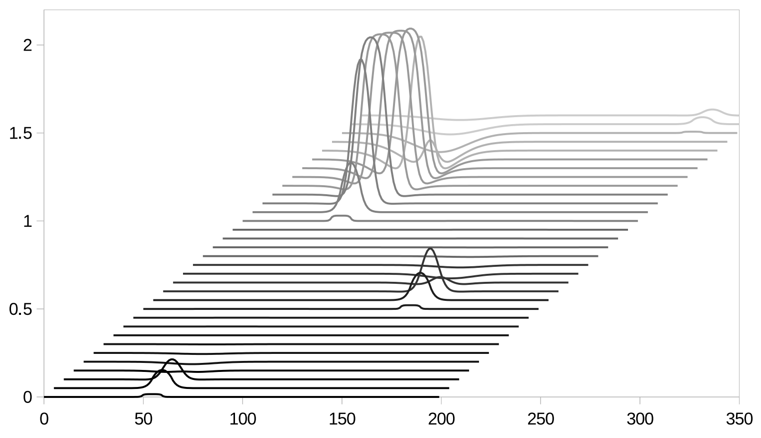

Figure 4.6: Noise in the FitzHugh-Nagumo Model
An example of noise in the FitzHugh-Nagumo Model. Small random perturbations. $D_v = 2$, $u_0 = 0.15$, $\epsilon_v = 0.2$. *The smaller perturbations quickly returns back to the homogenous state, whilst the larger ones excite for a moment. Compare this to 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.

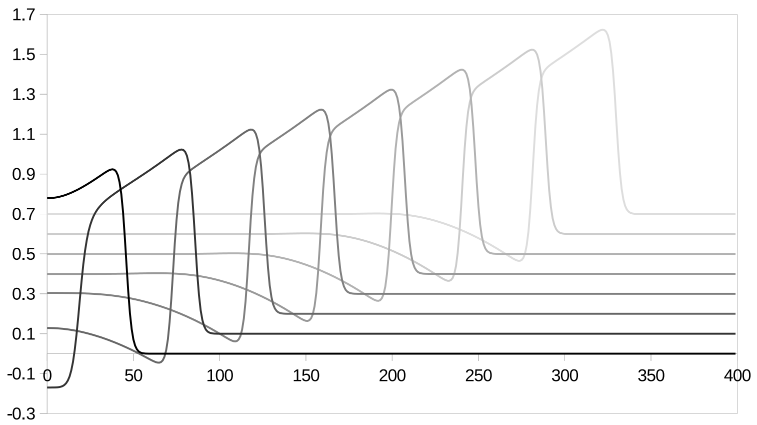
Figure 4.7: Propagating Wave in Radially-Symmetric 2D FitzHugh-Nagumo System
$D_v = 1$. *Profiles for greater values of time are shown in paler shades of grey.*

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

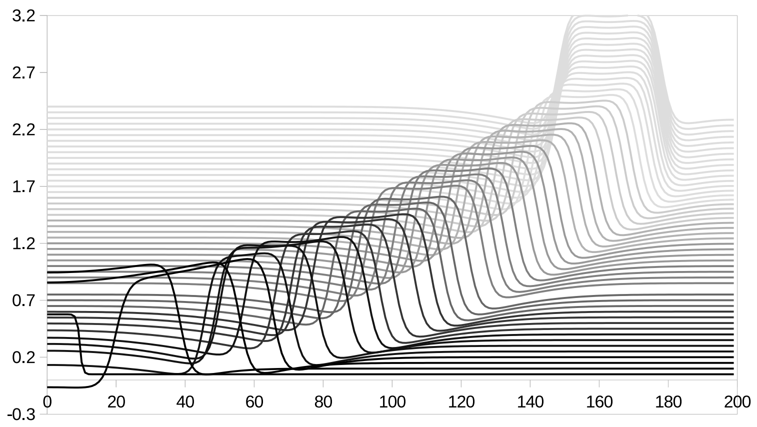
Figure 4.8: Stationary Annulus in Radially-Symmetric 2D FitzHugh-Nagumo System
$D_v = 4$. *Profiles for greater values of time are shown in paler shades of grey.*

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

Under certatin 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.

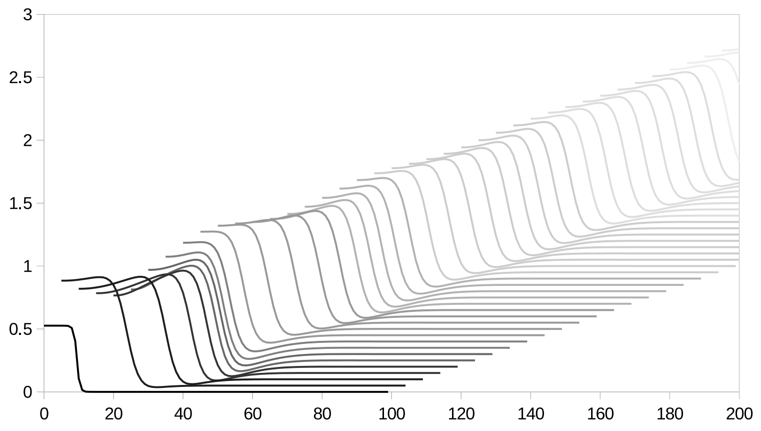

Figure 4.9: Stationary spot in Radially-Symmetric 2D FitzHugh-Nagumo System
$D_v = 2.8$, $\epsilon_v = 0.25$. *Profiles for greater values of time are shown in paler shades of grey.*

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 progressions 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.

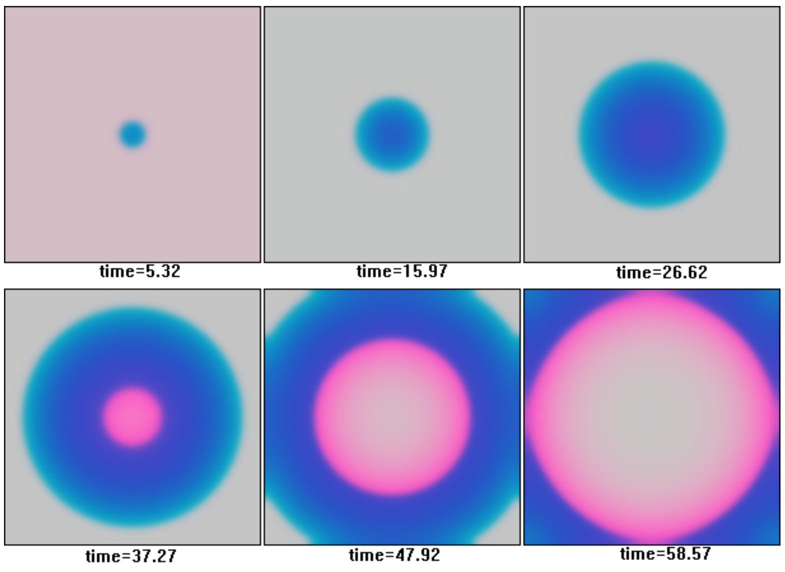

Figure 4.10: Two-dimensional FitzHugh-Nagumo Model - Travelling Wave
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 1$, which results in, as in the 1D-case, a travelling wave.

As $D_v \to 3$ 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.

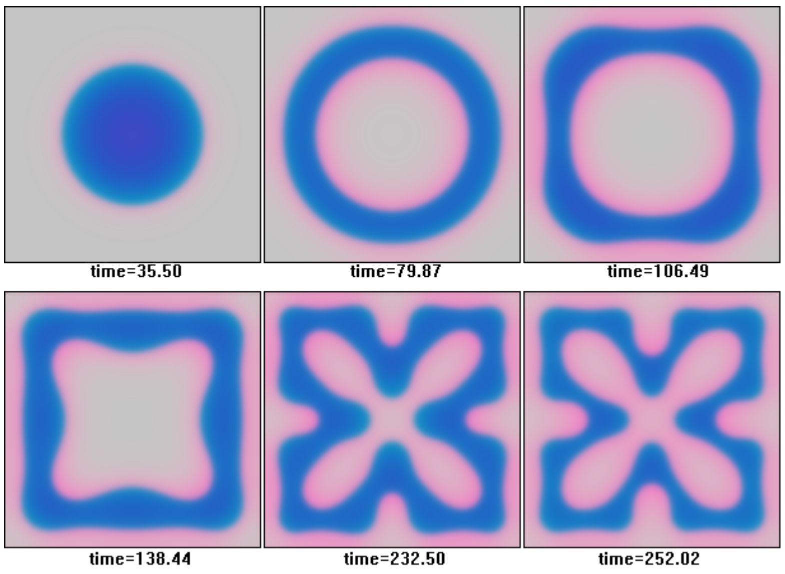

Figure 4.11: Two-dimensional FitzHugh-Nagumo Model - Oscillating Pattern
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 3$, which results in a pattern that oscillates between the fifth and sixth frames.

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

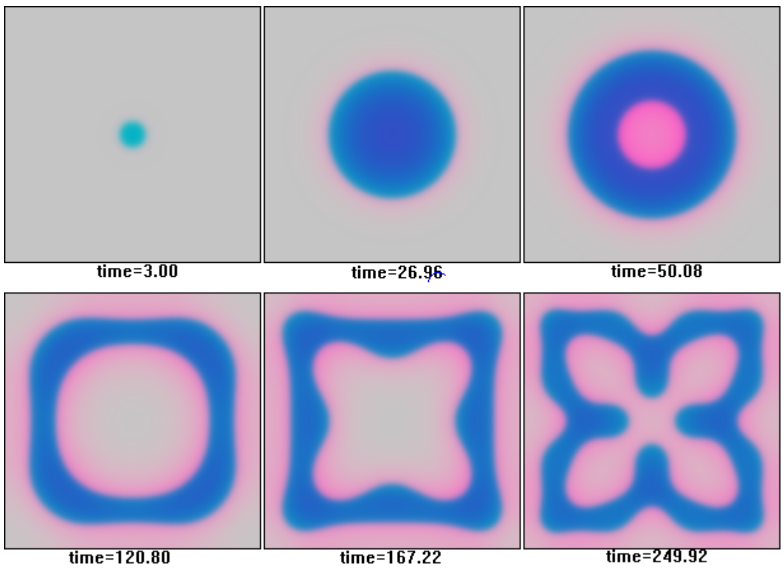

Figure 4.12: Two-dimensional FitzHugh-Nagumo Model - Stationary Pattern
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 4$, which results in, as in the 1D-case, a stationary 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.

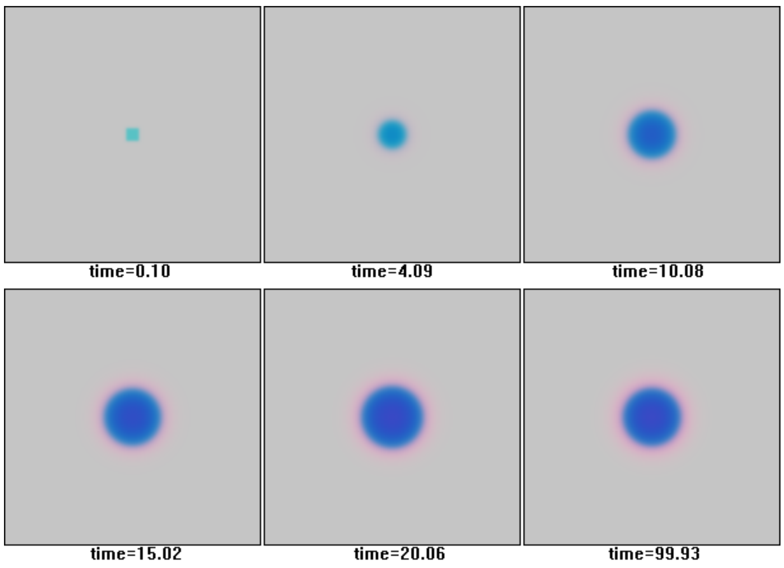

Figure 4.13: Two-dimensional FitzHugh-Nagumo Model - Stationary Spot
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 2.8$, $\epsilon_v = 0.25$ *which results in a stationary 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.

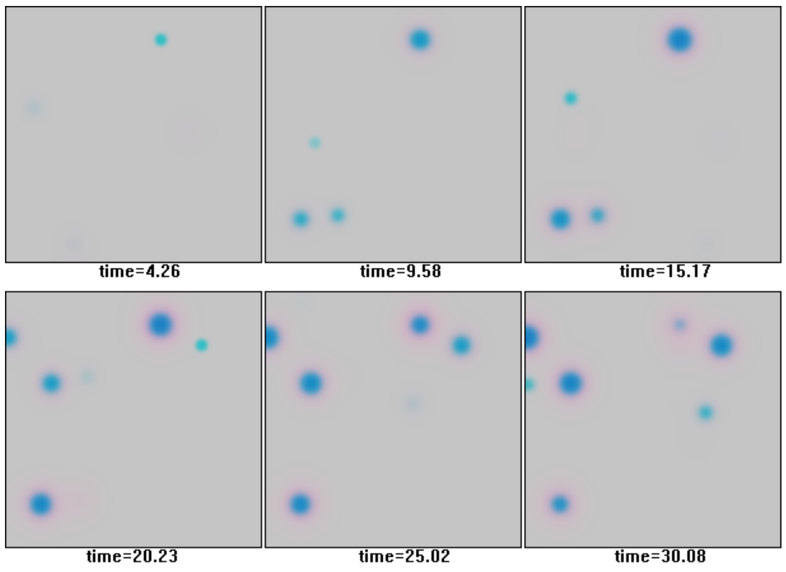

Figure 4.14: Two-dimensional FitzHugh-Nagumo Model with noise
Intensity of the blue and red values of each pixel are proportional to the concentrations of the activator and the inhibitor respectively. Medium size is 200 x 200. Here $D_v = 2$, $u_0 = 0.15$, $\epsilon_v = 0.2$. *Again, small perturbations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

As in the 1D case, for sufficently 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

```
// 1DFHN.cpp : implementation file
//

#include "stdafx.h"
#include "ReactionDiffusion.h"
#include "ReactionDiffusionDlg.h"
#include <math.h>
#include <afxwin.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

void CReactionDiffusionDlg::OnPaint()
{
    CPaintDC olddc(this); // device context for painting
    CDC dc; //We will bitblit this to olddc
    CBitmap bmpdc;

    if (IsIconic())
    {
    ...
    }
    else
    {
        CDialog::OnPaint();

        //Get plot area and save dimensions
        CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
        CRect R; pWnd->GetWindowRect(&R);
        INT nPlotWidth = R.Width();
        INT nPlotHeight = R.Height();

        ScreenToClient(&R);
        CPoint r,s;
        CPoint origin;
        LPCRECT plot_area(R);

        dc.CreateCompatibleDC(&dc);
        bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.Height());
        dc.SelectObject(&bmpdc);

        double xi,xx1,xx2,yy1,yy2,x_unit,y_unit;
        CString str;

        const int MEDIUM_SIZE = 200; //Size of medium

        bool dump = false; //Dump each frame to a tif file?

        const double Du = 1.0; //Diffusion coefficient of u
        double Dv = 1.0; //Diffusion coefficient of v
        double eu = 1.0; //Rate of kinetics of u
        double ev = 0.1; //Rate of kinetics of v
        double ku = 4.5;
        double u1 = 1;
        double u0 = 0.05;

        double ju0 = 0.0; //boundary flux of u at i=0
        double ju1 = 0.0; //boundary flux of u at i=MEDIUM_SIZE-1
        double jv0 = 0.0; //boundary flux of v at i=0
        double jv1 = 0.0; //boundary flux of v at i=MEDIUM_SIZE-1

        double u[MEDIUM_SIZE];
        double v[MEDIUM_SIZE];
```

```

    //Arrays for storing concentrations at next time step
    double u_next[MEDIUM_SIZE];
    double v_next[MEDIUM_SIZE];

    double delta_x=0.4; //space step
    double delta_x_square = delta_x*delta_x; //squaring to save computation
    double delta_t; //time step

    //Define delta_t in relation to Dv to ensure stability
    if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
    else delta_t=0.005;

    double current_time = 0.0;
    double final_time = 300;

    int current_step=0;
    int draw_interval=1; //Interval between drawing steps
    int offset = 0; //First timestep to draw

    //Index for for-loops
    int i;

    //Number of segments on each axis
    int x_segments=10;
    int y_segments=10;

    //Min and max for y axis
    double y_max = 1;
    double y_min = -0.2;

    //Interval between segments
    double x_interval = MEDIUM_SIZE/x_segments;
    double y_interval = (y_max-y_min)/y_segments;

    //Du,Dv divided by delta_x^2 to save time
    double Du_div = Du/(delta_x*delta_x);
```

```
        double Dv_div = Dv/(delta_x*delta_x);

        //Setting up initial conditions
        for (i=0;i<MEDIUM_SIZE;i++)
        {
            u[i]=0.1;
            v[i]=0.1;
        }

        for (i=MEDIUM_SIZE/2 - 5;i<MEDIUM_SIZE/2 + 5;i++)
        {
            u[i]=0.5;
        }

        while(current_time<final_time)
        {
            //Flux at boundaries
            u[0] = u[1] + ju0*delta_x;
            v[0] = v[1] + jv0*delta_x;
            u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*delta_x;
            v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*delta_x;

            //Explicit Euler method
            for(i=1;i<MEDIUM_SIZE-1;i++)
            {
                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]));
                v_next[i] = v[i] + delta_t*(Dv_div*(v[i+1]-2*v[i]+v[i-1]) + ev*(u[i] - v[i]));
            }

            //Draw curves every "draw_interval"-steps
            if((current_step-offset)%draw_interval==0)
            {

                //Clear plot area
                dc.FillRect(plot_area,WHITE_BRUSH);

                //Select blackPen
                dc.SelectObject(blackPen);

                xx1=R.TopLeft().x + 0.05*R.Width();
                yy1=R.BottomRight().y - 0.05*R.Height();
                xx2=R.BottomRight().x - 0.05*R.Width();
                yy2=R.TopLeft().y + 0.05*R.Height();

                double x_unit = (xx2-xx1)/MEDIUM_SIZE;
                double y_unit = (yy2-yy1)/(y_max-y_min);

                origin.x = xx1;
                origin.y = yy1 - y_min*y_unit;

                //Draw x-axis
                if(y_min>=0)
                {
                    dc.MoveTo(xx1,yy1);
                    dc.LineTo(xx2,yy1);
                }
                else
                {
                    dc.MoveTo(xx1,origin.y);
                    dc.LineTo(xx2,origin.y);
                }

                //Draw y-axis
                dc.MoveTo(xx1,yy1);
                dc.LineTo(xx1,yy2);

                //Label x-axis
                if(y_min>=0)
                {
                    s.x=xx1;
                    s.y=yy1;
                }
                else
                {
                    s.x=origin.x;
                    s.y=origin.y;
                }
                for(i=0;i<=x_segments;i++)
                {
                    r.y = s.y - 0.01*R.Height();
                    r.x = s.x + i*(xx2-xx1)/x_segments;
                    dc.MoveTo(r.x,r.y);
                    r.y = s.y + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.f"),x_interval*i);
                    dc.TextOut(r.x-10,r.y+1,str);
                }

                //Label y-axis
                for(i=0;i<=y_segments;i++)
                {
                    r.x = xx1 - 0.01*R.Height();
                    r.y = yy1 + i*(yy2-yy1)/y_segments;
                    dc.MoveTo(r.x,r.y);
                    r.x = xx1 + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),y_min +
    y_interval*i);
                    dc.TextOut(r.x-50,r.y-7,str);
                }

                str.Format(_T("time=%0.2f"),current_time);
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
    +30,str);
                str.Format(_T("activator␣-␣blue"));
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
    +60,str);
                str.Format(_T("inhibitor␣-␣red"));
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y
    +75,str);

                //Draw curve for u
                dc.SelectObject(bluePen);
                xi=(float)0;
                r.x=origin.x+xi*x_unit;
                r.y=origin.y+u[0]*y_unit;
                dc.MoveTo(r.x,r.y);

```

```
                for (i=0;i<MEDIUM_SIZE-1;i++)
                {
                    xi=(float)i;
                    r.x=origin.x+xi*x_unit;
                    r.y=origin.y+u[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                //Draw curve for v
                dc.SelectObject(redPen);
                xi=(float)0;
                r.x=origin.x+xi*x_unit;
                r.y=origin.y+v[0]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (i=1;i<MEDIUM_SIZE;i++)
                {
                    xi=(float)i;
                    r.x=origin.x+xi*x_unit;
                    r.y=origin.y+v[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                //Bit-blit to display
                olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
    Width(),R.Height(), &dc,R.TopLeft().x, R.TopLeft().y,
    SRCCOPY);
                //Dump frame to tif file
                if(dump) DumpToFile(R.Width(),R.Height(),R.
    TopLeft().x, R.TopLeft().y,&dc, current_step/
    draw_interval);

                //Dump values to log: Tab-separated values
                //                    Time steps on separate
     lines
                FILE * pFile;
                char fname[32];
                sprintf(fname,"log\\log.txt");

                if((current_step-offset)/draw_interval == 0)
     pFile = fopen (fname, "wb");
```

```
                    else pFile = fopen (fname, "ab");

                    for (i=0;i<MEDIUM_SIZE;i++)
                    {
                        fprintf (pFile, "%f\t",u[i]);
                    }
                    fprintf (pFile, "\n");
                    //for (i=0;i<MEDIUM_SIZE;i++)
                    //{
                    //    fprintf (pFile, "%f\t",v[i]);
                    //}
                    //fprintf (pFile, "\n");
                    fclose (pFile);
                }

                for (i=0;i<MEDIUM_SIZE;i++)
                {
                        u[i]=u_next[i];
                        v[i]=v_next[i];
                }

                //Increment
                current_time+=delta_t;
                current_step++;
            }
        }
}

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

One-dimensional Fitzhugh-Nagumo Model with noise

```
// 1DFHN-with-noise.cpp : implementation file
//

#include "stdafx.h"
#include "ReactionDiffusion.h"
```

```
#include "ReactionDiffusionDlg.h"
#include <math.h>
#include <afxwin.h>
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

void CReactionDiffusionDlg::OnPaint()
{
    CPaintDC olddc(this); // device context for painting
    CDC dc; //We will bitblit this to olddc
    CBitmap bmpdc;

    if (IsIconic())
    {
    ...
    }
    else
    {
        CDialog::OnPaint();

        //Get plot area and save dimensions
        CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
        CRect R; pWnd->GetWindowRect(&R);
        INT nPlotWidth = R.Width();
        INT nPlotHeight = R.Height();

        ScreenToClient(&R);
        CPoint r,s;
        CPoint origin;
        LPCRECT plot_area(R);

        dc.CreateCompatibleDC(&dc);
        bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
   Height());
        dc.SelectObject(&bmpdc);

        double xi,xx1,xx2,yy1,yy2,x_unit,y_unit;
        CString str;

        const int MEDIUM_SIZE = 200; //Size of medium

        bool dump = false; //Dump each frame to a tif file?

        const double Du = 1.0; //Diffusion coefficient of u
        double Dv = 2.0; //Diffusion coefficient of v
        double eu = 1.0; //Rate of kinetics of u
        double ev = 0.2; //Rate of kinetics of v
        double ku = 4.5;
        double u1 = 1;
        double u0 = 0.15;

        double stim,a,init;

        //Seed for RNG
        int seed = 123;

        double ju0 = 0.0; //boundary flux of u at i=0
        double ju1 = 0.0; //boundary flux of u at i=
    MEDIUM_SIZE-1
        double jv0 = 0.0; //boundary flux of v at i=0
        double jv1 = 0.0; //boundary flux of v at i=
    MEDIUM_SIZE-1

        double u[MEDIUM_SIZE];
        double v[MEDIUM_SIZE];

        double u_next[MEDIUM_SIZE];
        double v_next[MEDIUM_SIZE];

        double delta_x=0.4; //space step
        double delta_x_square = delta_x*delta_x; //squaring
    to save computation later
        double delta_t; //time step

        //Define delta_t in relation to Dv to ensure
    stability
        if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
        else delta_t=0.005;


        double current_time = 0.0;
        double final_time = 300;

        int current_step=0;
        int draw_interval=1; //Interval between drawing steps
        int offset = 0; //First timestep to draw

        //Index for for-loops
        int i;

        //Number of segments on each axis
        int x_segments=10;
        int y_segments=10;

        //Min and max for y axis
        double y_max = 1;
        double y_min = -0.2;

        //Interval between segments
        double x_interval = MEDIUM_SIZE/x_segments;
        double y_interval = (y_max-y_min)/y_segments;

        //Du,Dv divided by delta_x^2 to save time
        double Du_div = Du/(delta_x*delta_x);
        double Dv_div = Dv/(delta_x*delta_x);

        //Setting up initial conditions
        for (i=0;i<MEDIUM_SIZE;i++)
        {
            u[i]=0.1;
            v[i]=0.1;
        }

        srand(seed);
        a=0;


        while(current_time<final_time)
        {
```

```c
        //Flux at boundaries
        u[0] = u[1] + ju0*delta_x;
        v[0] = v[1] + jv0*delta_x;
        u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*delta_x;
        v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*delta_x;
        stim = 0;

        //Apply random noise at random position every 100 timesteps
        if(current_step%500==0)
        {
            init = (double) rand();
            init = init/RAND_MAX;
            init = init*MEDIUM_SIZE;

            a = (double) rand();
            a = a/RAND_MAX;
            a = a - 0.5;
        }

        //Explicit Euler method
        for(i=1;i<MEDIUM_SIZE-1;i++)
        {
            if(current_step%500<100 && i>init-5 && i<init+5)
            {
                stim = a*0.4;
            }
            else
            {
                stim = 0;
            }

            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);
            v_next[i] = v[i] + delta_t*(Dv_div*(v[i+1]-2*v[i]+v[i-1]) + ev*(u[i] - v[i]));
        }
```

```

            //Draw curves every "draw_interval"-steps
            if((current_step-offset)%draw_interval==0)
            {
                //Clear plot area
                dc.FillRect(plot_area,WHITE_BRUSH);
                //Select blackPen
                dc.SelectObject(blackPen);

                xx1=R.TopLeft().x + 0.05*R.Width();
                yy1=R.BottomRight().y - 0.1*R.Height();
                xx2=R.BottomRight().x - 0.05*R.Width();
                yy2=R.TopLeft().y + 0.05*R.Height();

                double x_unit = (xx2-xx1)/MEDIUM_SIZE;
                double y_unit = (yy2-yy1)/(y_max-y_min);

                origin.x = xx1;
                origin.y = yy1 - y_min*y_unit;

                //Draw x-axis
                if(y_min>=0)
                {
                    dc.MoveTo(xx1,yy1);
                    dc.LineTo(xx2,yy1);
                }else
                {
                    dc.MoveTo(xx1,origin.y);
                    dc.LineTo(xx2,origin.y);
                }

                //Draw y-axis
                dc.MoveTo(xx1,yy1);
                dc.LineTo(xx1,yy2);

                //Label x-axis
                if(y_min>=0)
                {
                    s.x=xx1;
                    s.y=yy1;
                }
                else
                {
                    s.x=origin.x;
                    s.y=origin.y;
                }
                for(i=0;i<=x_segments;i++)
                {
                    r.y = s.y - 0.01*R.Height();
                    r.x = s.x + i*(xx2-xx1)/x_segments;
                    dc.MoveTo(r.x,r.y);
                    r.y = s.y + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),x_interval*i);
                    dc.TextOut(r.x-10,r.y+1,str);
                }

                //Label y-axis
                for(i=0;i<=y_segments;i++)
                {
                    r.x = xx1 - 0.01*R.Height();
                    r.y = yy1 + i*(yy2-yy1)/y_segments;
                    dc.MoveTo(r.x,r.y);
                    r.x = xx1 + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),y_min + y_interval*i);
                    dc.TextOut(r.x-50,r.y-7,str);
                }

                str.Format(_T("time=%0.2f"),current_time);
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y+30,str);
                str.Format(_T("activator - blue"));
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y+60,str);
                str.Format(_T("inhibitor - red"));
                dc.TextOut(R.TopLeft().x+70,R.TopLeft().y+75,str);

                //Draw curve for u
                dc.SelectObject(bluePen);
```

```
                xi=(float)0;
                r.x=origin.x+xi*x_unit;
                r.y=origin.y+u[0]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (i=0;i<MEDIUM_SIZE-1;i++)
                {
                    xi=(float)i;
                    r.x=origin.x+xi*x_unit;
                    r.y=origin.y+u[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                //Draw curve for v
                dc.SelectObject(redPen);
                xi=(float)0;
                r.x=origin.x+xi*x_unit;
                r.y=origin.y+v[0]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (i=1;i<MEDIUM_SIZE;i++)
                {
                    xi=(float)i;
                    r.x=origin.x+xi*x_unit;
                    r.y=origin.y+v[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }


                olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
    Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
    SRCCOPY);
                //Dump frame to tif file
                if(dump) DumpToFile(R.Width(),R.Height(),R.
    TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
    ;

                //Dump values to log: Tab-separated values
                //                    Time steps on separate
     lines
                FILE * pFile;
```

```
            char fname[32];
            sprintf(fname,"log\\log.txt");

            if((current_step-offset)/draw_interval == 0) pFile = fopen (fname, "wb");
            else pFile = fopen (fname, "ab");

            for (i=0;i<MEDIUM_SIZE;i++)
            {
               fprintf (pFile, "%f\t",u[i]);
            }
            fprintf (pFile, "\n");
            //for (i=0;i<MEDIUM_SIZE;i++)
            //{
            //    fprintf (pFile, "%f\t",v[i]);
            //}
            //fprintf (pFile, "\n");
            fclose (pFile);
         }

         for (i=0;i<MEDIUM_SIZE;i++)
         {
               u[i]=u_next[i];
               v[i]=v_next[i];
         }

         //Increment
         current_time+=delta_t;
         current_step++;
      }
   }
}

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

Radially-Symmetric Fitzhugh-Nagumo Model

```
// Radially-Symmetric-FHN.cpp : implementation file
//

#include "stdafx.h"
#include "ReactionDiffusion.h"
#include "ReactionDiffusionDlg.h"
#include <math.h>
#include <afxwin.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

void CReactionDiffusionDlg::OnPaint()
{
    CPaintDC olddc(this); // device context for painting
    CDC dc; //We will bitblit this to olddc
    CBitmap bmpdc;

    if (IsIconic())
    {
    ...
    }
    else
    {
        CDialog::OnPaint();

        //Get plot area and save dimensions
        CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
        CRect R; pWnd->GetWindowRect(&R);
        INT nPlotWidth = R.Width();
        INT nPlotHeight = R.Height();

        ScreenToClient(&R);
        CPoint r,s,plot_origin,graph_origin;
        LPCRECT plot_area(R);

        COLORREF pixel_colour = RGB(0,0,0);

        dc.CreateCompatibleDC(&dc);
        bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.Height());
        dc.SelectObject(&bmpdc);

        double plot_left,plot_right,graph_left,graph_right,top,bottom,x_unit,y_unit,border,xj;
        double activator,inhibitor,zero_level;
        CString str;

        const int MEDIUM_SIZE = 200; //Size of medium

        //Minimizing computation for later plotting of medium
        int M2 = 2*MEDIUM_SIZE;
        int Msq = MEDIUM_SIZE*MEDIUM_SIZE;

        bool dump = false; //Dump each frame to a tif file?

        const double Du = 1.0; //Diffusion coefficient of u
        double Dv = 1.0; //Diffusion coefficient of v
        double eu = 1.0; //Rate of kinetics of u
        double ev = 0.1; //Rate of kinetics of v
        double ku = 4.5;
        double u1 = 1;
        double u0 = 0.05;

        double ju0 = 0.0; //boundary flux of u at i=0
        double ju1 = 0.0; //boundary flux of u at i=MEDIUM_SIZE-1
        double jv0 = 0.0; //boundary flux of v at i=0
        double jv1 = 0.0; //boundary flux of v at i=MEDIUM_SIZE-1

        double u[MEDIUM_SIZE];
        double v[MEDIUM_SIZE];

        //Arrays for storing concentrations at next time step
        double u_next[MEDIUM_SIZE];
        double v_next[MEDIUM_SIZE];

        double delta_x=0.4; //space step
        double delta_x_square = delta_x*delta_x; //squaring to save computation later
        double delta_t; //time step

        //Define delta_t in relation to Dv to ensure stability
        if(Dv>1.09) delta_t =(delta_x*delta_x)/(2.01*Dv);
        else delta_t=0.005;

        double current_time = 0.0;
        double final_time = 500;

        int current_step=0;
        int draw_interval=10; //Interval between drawing steps
        int offset = 0; //First timestep to draw

        //Index for for-loops
        int i,j,ii,isq,distsq;

        //Number of segments on each axis
        int x_segments=10;
        int y_segments=10;
        double y_max = 1;
        double y_min = -0.3;

        //Interval between segments
        double x_interval = MEDIUM_SIZE/x_segments;
        double y_interval = (y_max-y_min)/y_segments;

        border = 0.05*min(R.Height(),R.Width());
        top = R.TopLeft().y + border;
        bottom = R.BottomRight().y - border;
        plot_left = R.TopLeft().x + border;
        plot_right = R.TopLeft().x + 0.5*R.Width() - border;
        graph_right = R.BottomRight().x - border;
        graph_left = R.TopLeft().x + 0.5*R.Width() + border;
        plot_origin.x = int(floor((plot_right - plot_left - 2*MEDIUM_SIZE)*0.5));
```

```
        plot_origin.y = int(floor((bottom - top - 2*
    MEDIUM_SIZE)*0.5));

        x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
        y_unit = (top-bottom)/(y_max-y_min);
        graph_origin.x = graph_left;
        graph_origin.y = bottom - y_min*y_unit;

        //Du,Dv divided by delta_x^2 to save time
        double Du_div = Du/(delta_x*delta_x);
        double Dv_div = Dv/(delta_x*delta_x);

        //Setting up initial conditions
        for (i=0;i<MEDIUM_SIZE;i++)
        {
            u[i]=0.1;
            v[i]=0.1;
        }

        //Perturb at centre of medium
        for (i=0;i<5;i++)
        {
            u[i]=0.5;
        }

        while(current_time<final_time)
        {

            //Flux at boundaries
            u[0] = u[1] + ju0*delta_x;
            v[0] = v[1] + jv0*delta_x;
            u[MEDIUM_SIZE-1] = u[MEDIUM_SIZE-2] + ju1*
    delta_x;
            v[MEDIUM_SIZE-1] = v[MEDIUM_SIZE-2] + jv1*
    delta_x;

            //Explicit Euler method
            for(i=1;i<MEDIUM_SIZE-1;i++)
            {
                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]));
                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]));
            }

            //Draw curves every "draw_interval"-steps
            if((current_step-offset)%draw_interval==0)
            {
                y_interval = (y_max-y_min)/y_segments;
                y_unit = (top-bottom)/(y_max-y_min);

                //Clear plot area
                dc.FillRect(plot_area,WHITE_BRUSH); //Using
WHITE_BRUSH proper?

                //Plot on left
                //Draw bounding box
                dc.SelectObject(blackPen);
                dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
                dc.LineTo(plot_origin.x+2*MEDIUM_SIZE,
plot_origin.y-1);
                dc.LineTo(plot_origin.x+2*MEDIUM_SIZE,
plot_origin.y+2*MEDIUM_SIZE);
                dc.LineTo(plot_origin.x-1,plot_origin.y+2*
MEDIUM_SIZE);
                dc.LineTo(plot_origin.x-1,plot_origin.y-1);

                for(i=0;i<M2;i++)
                {
                    ii=i*M2;
                    isq = (i-MEDIUM_SIZE)*(i-MEDIUM_SIZE);
                    for(j=0;j<M2;j++)
                    {
                        distsq = isq + (j-MEDIUM_SIZE)*(j-
MEDIUM_SIZE);
                        if(distsq <= Msq)
                        {
                            int loc = (int) floor(sqrt( (
double)distsq ));
                            activator = u[loc];
                            inhibitor = v[loc];
                            activator = int((activator-y_min)*255/(y_max-y_min));
                            inhibitor = int((inhibitor-y_min)*255/(y_max-y_min));
                            zero_level = int((0-y_min)*255/(y_max-y_min));
                            pixel_colour = RGB(255-activator,255-inhibitor,255-zero_level);
                        }
                        else
                        {
                            pixel_colour = RGB(0,0,0);
                        }
                        dc.SetPixel(plot_origin.x+j, plot_origin.y+i, pixel_colour);
                    }
                }

                //Plot graph of cross section at right
                //Select blackPen
                dc.SelectObject(blackPen);

                //Draw x-axis
                if(y_min>=0)
                {
                    dc.MoveTo(graph_left,bottom);
                    dc.LineTo(graph_right,bottom);
                }
                else
                {
                    dc.MoveTo(graph_left,graph_origin.y);
                    dc.LineTo(graph_right,graph_origin.y);
                }

                //Draw y-axis
                dc.MoveTo(graph_left,bottom);
                dc.LineTo(graph_left,top);

                //Label x-axis
                if(y_min>=0)
                {
                    s.x=graph_origin.x;
                    s.y=bottom;
                }
                else
                {
                    s.x=graph_origin.x;
                    s.y=graph_origin.y;
                }
                for(j=0;j<=x_segments;j++)
                {
                    r.y = s.y - 0.01*R.Height();
                    r.x = s.x + j*(graph_right-graph_left)/
    x_segments;
                    dc.MoveTo(r.x,r.y);
                    r.y = s.y + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),x_interval*j);
                    dc.TextOut(r.x-10,r.y+1,str);
                }

                //Label y-axis
                for(j=0;j<=y_segments;j++)
                {
                    r.x = graph_left - 0.01*R.Height();
                    r.y = bottom + j*(top-bottom)/y_segments
    ;
                    dc.MoveTo(r.x,r.y);
                    r.x = graph_left + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),y_min +
    y_interval*j);
                    dc.TextOut(r.x-50,r.y-7,str);
                }

                str.Format(_T("time=%0.2f"),current_time);
                dc.TextOut(graph_left+70,top+30,str);
                str.Format(_T("activator␣-␣blue"));
                dc.TextOut(graph_left+70,top+60,str);
                str.Format(_T("inhibitor␣-␣red"));
                dc.TextOut(graph_left+70,top+75,str);

                i=0;
                dc.SelectObject(bluePen);
                xj=(float)0;
                r.x = graph_origin.x + xj*x_unit;
                r.y = graph_origin.y + u[i]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (i=1;i<MEDIUM_SIZE;i++)
                {
                    xj=(float)i;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+u[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                i=0;
                dc.SelectObject(redPen);
                xj=(float)0;
                r.x=graph_origin.x+xj*x_unit;
                r.y=graph_origin.y+v[i]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (i=i;i<MEDIUM_SIZE;i++)
                {
                    xj=(float)i;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+v[i]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,SRCCOPY);
                //Dump frame to tif file
                if(dump) DumpToFile(R.Width(),R.Height(),R.TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval);

                //Dump values to log: Tab-separated values
                //                      Time steps on separate lines
                FILE * pFile;
                char fname[32];
                sprintf(fname,"log\\log.txt");

                if((current_step-offset)/draw_interval == 0)
    pFile = fopen (fname, "wb");
                else pFile = fopen (fname, "ab");

                for (i=0;i<MEDIUM_SIZE;i++)
                {
                    fprintf (pFile, "%f\t",u[i]);
                }
                fprintf (pFile, "\n");
                //for (i=0;i<MEDIUM_SIZE;i++)
                //{
                //    fprintf (pFile, "%f\t",v[i]);
                //}
                //fprintf (pFile, "\n");
                fclose (pFile);
            }

            for (i=0;i<MEDIUM_SIZE;i++)
            {
                u[i]=u_next[i];
                v[i]=v_next[i];
            }

            //Increment
            current_time+=delta_t;
            current_step++;
        }
    }
}

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

Two-dimensional Fitzhugh-Nagumo Model

```
// 2DFHN.cpp : implementation file
//

#include "stdafx.h"
#include "ReactionDiffusion.h"
#include "ReactionDiffusionDlg.h"
#include <math.h>
#include <afxwin.h>
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

void CReactionDiffusionDlg::OnPaint()
{
    CPaintDC olddc(this); // device context for painting
    CDC dc; //We will bitblit this to olddc
    CBitmap bmpdc;

    if (IsIconic())
    {
    ...
    }
    else
    {
        CDialog::OnPaint();

        //Get plot area and save dimensions
        CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
        CRect R; pWnd->GetWindowRect(&R);
        INT nPlotWidth = R.Width();
        INT nPlotHeight = R.Height();

        ScreenToClient(&R);
        CPoint r,s,plot_origin,graph_origin;
        LPCRECT plot_area(R);

        COLORREF pixel_colour = RGB(0,0,0);

        dc.CreateCompatibleDC(&dc);
        bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
   Height());
        dc.SelectObject(&bmpdc);

        double plot_left,plot_right,graph_left,graph_right,
   top,bottom,x_unit,y_unit,border,xj;
        CString str;

        const int MEDIUM_SIZE = 200; //Size of medium:
   MEDIUM_SIZE * MEDIUM_SIZE

        bool dump = false; //Dump each frame to a tif file?

        const double Du = 1.0; //Diffusion coefficient of u
        double Dv = 1.0; //Diffusion coefficient of v
        double eu = 1.0; //Rate of kinetics of u
        double ev = 0.1; //Rate of kinetics of v
        double ku = 4.5;
        double u1 = 1;
        double u0 = 0.05;

        double stim=0.;

        double ju0 = 0.0; //boundary flux of u at i=0
        double ju1 = 0.0; //boundary flux of u at i=
   MEDIUM_SIZE-1

        double jv0 = 0.0; //boundary flux of v at i=0
        double jv1 = 0.0; //boundary flux of v at i=
   MEDIUM_SIZE-1

        int vector_size = MEDIUM_SIZE*MEDIUM_SIZE;

        double* u;
        double* v;
        double* u_next;
        double* v_next;

        u = new double [vector_size];
        v = new double [vector_size];
```

```
        u_next = new double [vector_size];
        v_next = new double [vector_size];

        double delta_x=0.4; //space step
        double delta_x_square = delta_x*delta_x; //squaring to save computation later
        double delta_t; //time step

        //Define delta_t in relation to Dv to ensure stability
        if(Dv>1.09) delta_t =(delta_x*delta_x)/(6.01*Dv);
        else delta_t=0.005;

        double current_time = 0.0;
        double final_time = 100;

        int current_step=0;
        int draw_interval=10; //Interval between drawing steps
        int offset = 0; //First timestep to draw

        double activator,inhibitor,zero_level;

        //Number of segments on each axis
        int x_segments=20;
        int y_segments=20;
        double y_max = 1;
        double y_min = -0.3;

        double u_max=y_max;
        double u_min=y_min;
        double v_max=y_max;
        double v_min=y_min;

        //Interval between segments
        double x_interval = MEDIUM_SIZE/x_segments;
        double y_interval = (y_max-y_min)/y_segments;

        //Index for for-loops
        int i,j,ii,ip1,im1;

        border = 0.05*min(R.Height(),R.Width());
        top = R.TopLeft().y + border;
        bottom = R.BottomRight().y - border;
        plot_left = R.TopLeft().x + border;
        plot_right = R.TopLeft().x + 0.5*R.Width() - border;
        graph_right = R.BottomRight().x - border;
        graph_left = R.TopLeft().x + 0.5*R.Width() + border;
        plot_origin.x = int(floor((plot_right - plot_left - MEDIUM_SIZE)*0.5));
        plot_origin.y = int(floor((bottom - top - MEDIUM_SIZE)*0.5));

        x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
        y_unit = (top-bottom)/(y_max-y_min);
        graph_origin.x = graph_left;
        graph_origin.y = bottom - y_min*y_unit;

        //Du,Dv divided by delta_x^2 to save time
        double Du_div = Du/(delta_x*delta_x*6);
        double Dv_div = Dv/(delta_x*delta_x*6);

        //Setting up initial conditions
        for (i=0;i<MEDIUM_SIZE;i++)
        {
            ii=i*MEDIUM_SIZE;
            for (j=0;j<MEDIUM_SIZE;j++)
            {
                u[ii + j]=0.0;
                v[ii + j]=0.0;
            }
        }

        for (i=(MEDIUM_SIZE/2 - 5);i<(MEDIUM_SIZE/2 + 5);i++)
        {
            ii=i*MEDIUM_SIZE;
            for (j=(MEDIUM_SIZE/2 - 5);j<(MEDIUM_SIZE/2 + 5);j++)
            {
                u[ii + j]=0.5;
            }
        }

        while(current_time<final_time)
        {
            //Flux at boundaries
            ii = MEDIUM_SIZE*MEDIUM_SIZE;
            for(j=0;j<MEDIUM_SIZE;j++)
            {
                u[0 + j] = u[MEDIUM_SIZE + j];
                u[ii-MEDIUM_SIZE + j] = u[ii-2*MEDIUM_SIZE + j];
                v[0 + j] = v[MEDIUM_SIZE + j];
                v[ii-MEDIUM_SIZE + j] = v[ii-2*MEDIUM_SIZE + j];
            }
            for(i=0;i<MEDIUM_SIZE;i++)
            {
                ii=i*MEDIUM_SIZE;
                u[ii + 0] = u[ii + 1];
                u[ii + MEDIUM_SIZE-1] = u[ii + MEDIUM_SIZE-2];
                v[ii + 0] = v[ii + 1];
                v[ii + MEDIUM_SIZE-1] = v[ii + MEDIUM_SIZE-2];
            }

            //Explicit Euler Method
            for(i=1;i<MEDIUM_SIZE-1;i++)
            {
                ii=i*MEDIUM_SIZE;
                ip1=(i+1)*MEDIUM_SIZE;
                im1=(i-1)*MEDIUM_SIZE;
                for(j=1;j<MEDIUM_SIZE-1;j++)
                {
                    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]));
                    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])
);
                }
            }

            for (i=1;i<MEDIUM_SIZE-1;i++)
            {
                ii=i*MEDIUM_SIZE;
                for (j=1;j<MEDIUM_SIZE-1;j++)
                {
                    u[ii + j]=u_next[ii + j];
                    v[ii + j]=v_next[ii + j];
                }
            }

            //Draw curves every "draw_interval"-steps
            if(current_step%draw_interval==0)
            {
                y_interval = (y_max-y_min)/y_segments;
                y_unit = (top-bottom)/(y_max-y_min);

                //Clear plot area
                dc.FillRect(plot_area,WHITE_BRUSH);

                //Plot on left
                //Draw bounding box
                dc.SelectObject(blackPen);
                dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
                dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y-1);
                dc.LineTo(plot_origin.x+MEDIUM_SIZE,
plot_origin.y+MEDIUM_SIZE);
                dc.LineTo(plot_origin.x-1,plot_origin.y+
MEDIUM_SIZE);
                dc.LineTo(plot_origin.x-1,plot_origin.y-1);

                for(i=0;i<MEDIUM_SIZE;i++)
                {
                    ii=i*MEDIUM_SIZE;
                    for(j=0;j<MEDIUM_SIZE;j++)
                    {
                        activator = u[ii + j];
                        inhibitor = v[ii + j];

                        activator = int((activator-y_min)*255/(y_max-y_min));
                        inhibitor = int((inhibitor-y_min)*255/(y_max-y_min));
                        zero_level = int((0-y_min)*255/(y_max-y_min));
                        pixel_colour = RGB(255-activator,255-inhibitor,255-zero_level);
                        dc.SetPixel(plot_origin.x+j, plot_origin.y+i, pixel_colour);
                    }
                }

                //Plot graph of centre cross section at right
                //Select blackPen
                dc.SelectObject(blackPen);

                //Draw x-axis
                //centre row (this won't change)
                ii = int(MEDIUM_SIZE*MEDIUM_SIZE*0.5);
                if(y_min>=0)
                {
                    dc.MoveTo(graph_left,bottom);
                    dc.LineTo(graph_right,bottom);
                }
                else
                {
                    dc.MoveTo(graph_left,graph_origin.y);
                    dc.LineTo(graph_right,graph_origin.y);
                }

                //Draw y-axis
                dc.MoveTo(graph_left,bottom);
                dc.LineTo(graph_left,top);

                //Label x-axis
```

```
if(y_min>=0)
{
    s.x=graph_origin.x;
    s.y=bottom;
}
else
{
    s.x=graph_origin.x;
    s.y=graph_origin.y;
}
for(j=0;j<=x_segments;j++)
{
    r.y = s.y - 0.01*R.Height();
    r.x = s.x + j*(graph_right-graph_left)/x_segments;
    dc.MoveTo(r.x,r.y);
    r.y = s.y + 0.01*R.Height();
    dc.LineTo(r.x,r.y);
    str.Format(_T("%0.2f"),x_interval*j);
    dc.TextOut(r.x-10,r.y+1,str);
}

//Label y-axis
for(j=0;j<=y_segments;j++)
{
    r.x = graph_left - 0.01*R.Height();
    r.y = bottom + j*(top-bottom)/y_segments;
    dc.MoveTo(r.x,r.y);
    r.x = graph_left + 0.01*R.Height();
    dc.LineTo(r.x,r.y);
    str.Format(_T("%0.2f"),y_min + y_interval*j);
    dc.TextOut(r.x-50,r.y-7,str);
}

str.Format(_T("time=%0.2f"),current_time);
dc.TextOut(graph_left+70,top+30,str);
str.Format(_T("activator␣-␣blue"));
dc.TextOut(graph_left+70,top+60,str);
str.Format(_T("inhibitor␣-␣red"));
```

```
                dc.TextOut(graph_left+70,top+75,str);

                j=0;
                dc.SelectObject(bluePen);
                xj=(float)0;
                r.x = graph_origin.x + xj*x_unit;
                r.y = graph_origin.y + u[ii+j]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (j=1;j<MEDIUM_SIZE;j++)
                {
                    xj=(float)j;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+u[ii+j]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                j=0;
                dc.SelectObject(redPen);
                xj=(float)0;
                r.x=graph_origin.x+xj*x_unit;
                r.y=graph_origin.y+v[ii+j]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (j=1;j<MEDIUM_SIZE;j++)
                {
                    xj=(float)j;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+v[ii+j]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.
Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,
SRCCOPY);
                //Dump frame to tif file
                if(dump) DumpToFile(R.Width(),R.Height(),R.
TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval)
;
            }

            //Increment
            current_time+=delta_t;
            current_step++;
        }
    }
}

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

Two-dimensional Fitzhugh-Nagumo Model with noise

```
// 2DFHN-with-noise.cpp : implementation file
//

#include "stdafx.h"
#include "ReactionDiffusion.h"
#include "ReactionDiffusionDlg.h"
#include <math.h>
#include <afxwin.h>
#include <stdlib.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

void CReactionDiffusionDlg::OnPaint()
{
    CPaintDC olddc(this); // device context for painting
    CDC dc; //We will bitblit this to olddc
    CBitmap bmpdc;

    if (IsIconic())
    {
    ...
    }
    else
    {
        CDialog::OnPaint();

        //Get plot area and save dimensions
        CWnd* pWnd = GetDlgItem(IDC_PLOT_AREA);
        CRect R; pWnd->GetWindowRect(&R);
        INT nPlotWidth = R.Width();
        INT nPlotHeight = R.Height();

        ScreenToClient(&R);
        CPoint r,s,plot_origin,graph_origin;
        LPCRECT plot_area(R);

        COLORREF pixel_colour = RGB(0,0,0);

        dc.CreateCompatibleDC(&dc);
        bmpdc.CreateCompatibleBitmap(&olddc, R.Width(), R.
    Height());
        dc.SelectObject(&bmpdc);

        double plot_left,plot_right,graph_left,graph_right,
    top,bottom,x_unit,y_unit,border,xj;
        CString str;

        const int MEDIUM_SIZE = 200; //Size of medium:
    MEDIUM_SIZE * MEDIUM_SIZE

        bool dump = false; //Dump each frame to a tif file?

        const double Du = 1.0; //Diffusion coefficient of u
        double Dv = 2.0; //Diffusion coefficient of v
        double eu = 1.0; //Rate of kinetics of u
        double ev = 0.2; //Rate of kinetics of v
        double ku = 4.5;
        double u1 = 1;
        double u0 = 0.05;

        double stim,a,init;

        //Seed for RNG
        int seed = 123;

        double ju0 = 0.0; //boundary flux of u at i=0
        double ju1 = 0.0; //boundary flux of u at i=
   MEDIUM_SIZE-1

        double jv0 = 0.0; //boundary flux of v at i=0
        double jv1 = 0.0; //boundary flux of v at i=
   MEDIUM_SIZE-1

        int vector_size = MEDIUM_SIZE*MEDIUM_SIZE;

        double* u;
        double* v;
        double* u_next;
        double* v_next;

        u = new double [vector_size];
        v = new double [vector_size];
        u_next = new double [vector_size];
        v_next = new double [vector_size];

        double delta_x=0.4; //space step
        double delta_x_square = delta_x*delta_x; //squaring
   to save computation later
        double delta_t; //time step

        //Define delta_t in relation to Dv to ensure
   stability
        if(Dv>1.09) delta_t =(delta_x*delta_x)/(6.01*Dv);
        else delta_t=0.005;

        double current_time = 0.0;
        double final_time = 100;

        int current_step=0;
        int draw_interval=50; //Interval between drawing
   steps
        int offset = 0; //First timestep to draw

        double activator,inhibitor,zero_level;

        //Number of segments on each axis
```

```
        int x_segments=20;
        int y_segments=20;
        double y_max = 1;
        double y_min = -0.3;

        double u_max=y_max;
        double u_min=y_min;
        double v_max=y_max;
        double v_min=y_min;

        //Interval between segments
        double x_interval = MEDIUM_SIZE/x_segments;
        double y_interval = (y_max-y_min)/y_segments;

        //Index for for-loops
        int i,j,ii,ip1,im1;

        border = 0.05*min(R.Height(),R.Width());
        top = R.TopLeft().y + border;
        bottom = R.BottomRight().y - border;
        plot_left = R.TopLeft().x + border;
        plot_right = R.TopLeft().x + 0.5*R.Width() - border;
        graph_right = R.BottomRight().x - border;
        graph_left = R.TopLeft().x + 0.5*R.Width() + border;
        plot_origin.x = int(floor((plot_right - plot_left - MEDIUM_SIZE)*0.5));
        plot_origin.y = int(floor((bottom - top - MEDIUM_SIZE)*0.5));

        x_unit = (graph_right-graph_left)/MEDIUM_SIZE;
        y_unit = (top-bottom)/(y_max-y_min);
        graph_origin.x = graph_left;
        graph_origin.y = bottom - y_min*y_unit;

        //Du,Dv divided by delta_x^2 to save time
        double Du_div = Du/(delta_x*delta_x*6);
        double Dv_div = Dv/(delta_x*delta_x*6);

        //Setting up initial conditions
        for (i=0;i<MEDIUM_SIZE;i++)
        {
            ii=i*MEDIUM_SIZE;
            for (j=0;j<MEDIUM_SIZE;j++)
            {
                u[ii + j]=0.0;
                v[ii + j]=0.0;
            }
        }

        srand(seed);
        a=0;

        while(current_time<final_time)
        {
            //Apply noise every 100 time steps
            if(current_step%500==0){
                init = (double) rand();
                init = init/RAND_MAX;
                init = init*vector_size;

                init_coord.x = ((int) floor(init))%MEDIUM_SIZE;
                init_coord.y = floor(init/MEDIUM_SIZE);

                a = (double) rand();
                a = a/RAND_MAX;
                a = a - 0.5;
            }

            //Flux at boundaries
            ii = MEDIUM_SIZE*MEDIUM_SIZE;
            for(j=0;j<MEDIUM_SIZE;j++)
            {
                u[0 + j] = u[MEDIUM_SIZE + j];
                u[ii-MEDIUM_SIZE + j] = u[ii-2*MEDIUM_SIZE + j];
                v[0 + j] = v[MEDIUM_SIZE + j];
                v[ii-MEDIUM_SIZE + j] = v[ii-2*MEDIUM_SIZE + j];
            }
            for(i=0;i<MEDIUM_SIZE;i++)
            {
```

```
            ii=i*MEDIUM_SIZE;
            u[ii + 0] = u[ii + 1];
            u[ii + MEDIUM_SIZE-1] = u[ii + MEDIUM_SIZE-2];
            v[ii + 0] = v[ii + 1];
            v[ii + MEDIUM_SIZE-1] = v[ii + MEDIUM_SIZE-2];
        }

        //Explicit Euler Method
        for(i=1;i<MEDIUM_SIZE-1;i++)
        {
            ii=i*MEDIUM_SIZE;
            ip1=(i+1)*MEDIUM_SIZE;
            im1=(i-1)*MEDIUM_SIZE;
            for(j=1;j<MEDIUM_SIZE-1;j++)
            {
                if(current_step%500<100 && ((init_coord.x - i)*(init_coord.x - i) + (init_coord.y - j)*(init_coord.y - j))<25){
                stim = a*0.4;
                }else{
                stim = 0;
                }

                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);
                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]));
            }
        }

        for (i=1;i<MEDIUM_SIZE-1;i++)
        {
            ii=i*MEDIUM_SIZE;
            for (j=1;j<MEDIUM_SIZE-1;j++)
            {
                u[ii + j]=u_next[ii + j];
                v[ii + j]=v_next[ii + j];
            }
        }

        //Draw curves every "draw_interval"-steps
        if(current_step%draw_interval==0)
        {
            y_interval = (y_max-y_min)/y_segments;
            y_unit = (top-bottom)/(y_max-y_min);

            //Clear plot area
            dc.FillRect(plot_area,WHITE_BRUSH);

            //Plot on left
            //Draw bounding box
            dc.SelectObject(blackPen);
            dc.MoveTo(plot_origin.x-1,plot_origin.y-1);
            dc.LineTo(plot_origin.x+MEDIUM_SIZE,plot_origin.y-1);
            dc.LineTo(plot_origin.x+MEDIUM_SIZE,plot_origin.y+MEDIUM_SIZE);
            dc.LineTo(plot_origin.x-1,plot_origin.y+MEDIUM_SIZE);
            dc.LineTo(plot_origin.x-1,plot_origin.y-1);

            for(i=0;i<MEDIUM_SIZE;i++)
            {
                ii=i*MEDIUM_SIZE;
                for(j=0;j<MEDIUM_SIZE;j++)
                {
                    activator = u[ii + j];
                    inhibitor = v[ii + j];

                    activator = int((activator-y_min)*255/(y_max-y_min));
                    inhibitor = int((inhibitor-y_min)*255/(y_max-y_min));
                    zero_level = int((0-y_min)*255/(y_max-y_min));
                    pixel_colour = RGB(255-activator,255-inhibitor,255-zero_level);
                    dc.SetPixel(plot_origin.x+j, plot_origin.y+i, pixel_colour);
                }
            }

            //Plot graph of centre cross section at right
            //Select blackPen
            dc.SelectObject(blackPen);

            //Draw x-axis
            //centre row (this won't change)
            ii = int(MEDIUM_SIZE*MEDIUM_SIZE*0.5);
            if(y_min>=0)
            {
                dc.MoveTo(graph_left,bottom);
                dc.LineTo(graph_right,bottom);
            }
            else
            {
                dc.MoveTo(graph_left,graph_origin.y);
                dc.LineTo(graph_right,graph_origin.y);
            }

            //Draw y-axis
            dc.MoveTo(graph_left,bottom);
            dc.LineTo(graph_left,top);

            //Label x-axis
            if(y_min>=0)
            {
                s.x=graph_origin.x;
                s.y=bottom;
            }
            else
            {
                s.x=graph_origin.x;
```

```
                    s.y=graph_origin.y;
                }
                for(j=0;j<=x_segments;j++)
                {
                    r.y = s.y - 0.01*R.Height();
                    r.x = s.x + j*(graph_right-graph_left)/x_segments;
                    dc.MoveTo(r.x,r.y);
                    r.y = s.y + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),x_interval*j);
                    dc.TextOut(r.x-10,r.y+1,str);
                }

                //Label y-axis
                for(j=0;j<=y_segments;j++)
                {
                    r.x = graph_left - 0.01*R.Height();
                    r.y = bottom + j*(top-bottom)/y_segments;
                    dc.MoveTo(r.x,r.y);
                    r.x = graph_left + 0.01*R.Height();
                    dc.LineTo(r.x,r.y);
                    str.Format(_T("%0.2f"),y_min + y_interval*j);
                    dc.TextOut(r.x-50,r.y-7,str);
                }

                str.Format(_T("time=%0.2f"),current_time);
                dc.TextOut(graph_left+70,top+30,str);
                str.Format(_T("activator␣-␣blue"));
                dc.TextOut(graph_left+70,top+60,str);
                str.Format(_T("inhibitor␣-␣red"));
                dc.TextOut(graph_left+70,top+75,str);

                j=0;
                dc.SelectObject(bluePen);
                xj=(float)0;
                r.x = graph_origin.x + xj*x_unit;
                r.y = graph_origin.y + u[ii+j]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (j=1;j<MEDIUM_SIZE;j++)
                {
                    xj=(float)j;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+u[ii+j]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                j=0;
                dc.SelectObject(redPen);
                xj=(float)0;
                r.x=graph_origin.x+xj*x_unit;
                r.y=graph_origin.y+v[ii+j]*y_unit;
                dc.MoveTo(r.x,r.y);

                for (j=1;j<MEDIUM_SIZE;j++)
                {
                    xj=(float)j;
                    r.x=graph_origin.x+xj*x_unit;
                    r.y=graph_origin.y+v[ii+j]*y_unit;
                    dc.LineTo(r.x,r.y);
                }

                olddc.BitBlt(R.TopLeft().x, R.TopLeft().y,R.Width(),R.Height(),&dc,R.TopLeft().x, R.TopLeft().y,SRCCOPY);
                //Dump frame to tif file
                if(dump) DumpToFile(R.Width(),R.Height(),R.TopLeft().x,R.TopLeft().y,&dc,current_step/draw_interval);
            }

            //Increment
            current_time+=delta_t;
            current_step++;
        }
    }
}

```

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