東北大学
TOHOKU UNIVERSITY

TOHOKU UNIVERSITY
MATHEMATICAL MODELING AND COMPUTATION
FINAL REPORT

Turing's Morphogenesis Applied to Patterns in Animal Coats

Student
Elion Mehdi - B8TL1012

Department
Aerospace Engineering

TOHOKU
UNIVERSITY

August 2nd 2018

Table of Contents

1 Introduction

Mammals show miscellaneous fur patterns that are very complex and interesting. These patterns have been subjected to various zoological studies which noted the incredible variety of shapes and structures present within worldwide fauna. In 1952, in a founding article entitled "The Chemical Basis of Morphogenesis" [5], Alan Mathison Turing proposed a mathematical model according to which the association of molecular diffusion and some chemical reactions could lead to spatial structuration of chemical species. He suggested that these purely physicochemical phenomena could be the basis for morphogenesis in flora and fauna. Such structures are often mentioned as "reaction-diffusion" structures, in reference to the reaction-diffusion equations that are used to mathematically model these phenomena, or even "Turing structures". More specifically, although diffusion tends to smooth the different concentrations, the coupling with reactions between a species that activates its own production and the production of an inhibiting species, in addition to a sufficient gap between diffusion coefficients can lead to spontaneous formation of periodical patterns (dots or stripes) in an initially homogenous environment.

Indeed, forty years after Turing's work, the first experimental proof of Turing structures, evenly spaced spots or stripes, was shown (Castets et coll. 1990, Physical Review Letters, n° 64, p. 2953). Since then, a lot a work has been done based on Turing's model, which became a reference to explain pattern formation in biology. Notably, Murray, in his book *Mathematical Biology* [3], also studied pattern formation using reaction-diffusion models (that can be unstable) and explains fur patterns by one single phenomenon. Skin and hair pigmentation is due to melanin production. That production of melanin is due to gene pool and exposure to the environment (UV for humans, for instance). That model is based on the assumption that mammal's furs only reflect spatio-temporal disparities of morphogens in epidermis (for the color of skin and hair). Morphogens are molecules that are essential to cellular development because they enable the specification of different cell types, their localization and orientation. According to Murray, genetic predispositions create a sort of outline of spot shapes during embryonic stage, and then melanin, whose concentration is subjected to reaction-diffusion, scatters and diffuses to obtain the patterns observed in nature.

In that report, we will numerically solve such a reaction-diffusion model in order to obtain patterns similar to the ones that can be found in some animal coats.

2 Physical Model

For animal coat pattern formation, we will use a reaction diffusion model with two morphogens that are involved in production (or non-production) of melanin by melanocytes : oxygen substrate and uricase enzymes. They respectively act as activator and inhibitor in chemical reactions that rule melanin production.

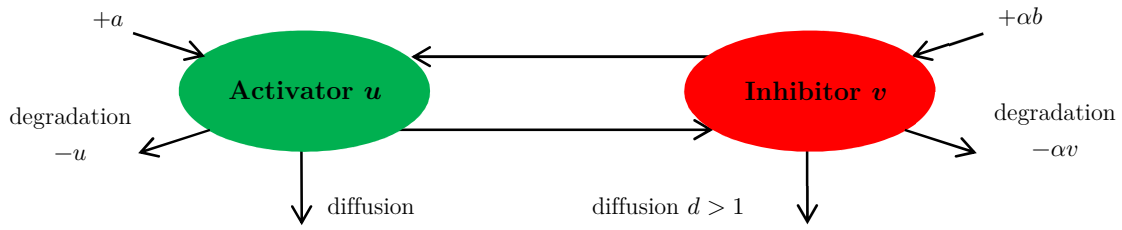
That non-dimensional 2D reaction-diffusion system can be written as follows :

$$\begin{cases} \dfrac{\partial u}{\partial t}(x,y,t) = \gamma f\big(u(x,y,t), v(x,y,t)\big) + \ \Delta u(x,y,t) \\ \dfrac{\partial v}{\partial t}(x,y,t) = \gamma g\big(u(x,y,t), v(x,y,t)\big) + d\Delta v(x,y,t) \\ u(x,y,0) = u_0(x,y) \qquad v(x,y,t) = v_0(x,y) \end{cases}$$

for $\quad x \in [x_{min}, x_{max}], \quad y \in [y_{min}, y_{max}], \quad t \in]0, T]$

with
- $f(u,v) = \ a - u - \frac{\rho u v}{1+u+Ku^2}$ is the reaction term of the equation on u
- $g(u,v) = \alpha(b-v) - \ \frac{\rho u v}{1+u+Ku^2}$ is the reaction term of the equation on v
- u is proportional to the concentration of oxygen substrate
- v is proportional to the concentration of uricase enzymes
- α, a, b, ρ are strictly positive parameters
- $d > 0$ is the relative diffusion coefficient
- $K > 0$ describes the magnitude of the inhibiting power of v
- $\gamma > 0$ is a positive parameter proportional to the squared area of the domain

The kinetics may be represented in a schematic way, as follows ;



In order to ensure that whatever happens to the system is due to "self-generation" rather than interaction with external environment, we will choose zero-flux boundary conditions (also called Neumann boundary condition) :

$$(\vec{n}.\nabla)\begin{pmatrix}u\\v\end{pmatrix} = \begin{pmatrix}0\\0\end{pmatrix} \quad on \quad \partial([x_{min}, x_{max}] \times [y_{min}, y_{max}])$$

That condition can be re-written :

$$\begin{cases} \left.\dfrac{\partial u}{\partial x}\right|_{x=x_{min}} = \left.\dfrac{\partial v}{\partial x}\right|_{x=x_{min}} = \left.\dfrac{\partial u}{\partial x}\right|_{x=x_{max}} = \left.\dfrac{\partial v}{\partial x}\right|_{x=x_{max}} = 0 \\ \left.\dfrac{\partial u}{\partial y}\right|_{x=y_{min}} = \left.\dfrac{\partial v}{\partial y}\right|_{x=y_{min}} = \left.\dfrac{\partial u}{\partial y}\right|_{x=y_{max}} = \left.\dfrac{\partial v}{\partial y}\right|_{x=y_{max}} = 0 \end{cases}$$

3 Mathematical Model

3.1 Discretization of Spatio-Temporal Domain

To solve that system on a finite spatio-temporal domain $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [0, T]$, we first need to discretize that domain. To do so, we define a number of points for the x axis (N_x), for the y axis (N_y) and for time (N_t) and proceed as follows :

$$\begin{cases} x_i = x_{min} + i\Delta x & for\ i \in [|0, N_x + 1|] \\ y_j = y_{min} + j\Delta y & for\ j \in [|0, N_y + 1|] \\ t_n = n\Delta t & for\ n \in [|0, N_t|] \end{cases}$$

where • $\Delta x = \frac{x_{max} - x_{min}}{N_x + 1}$ • $\Delta y = \frac{y_{max} - y_{min}}{N_y + 1}$ • $\Delta t = \frac{T}{N_t}$

Numerical Approximation

Given (u, v) the exact solution of our problem, let ($u^n_{i,j}$, $v^n_{i,j}$) be the approximation of $\Big(u(x_i, y_j, t_n),\ v(x_i, y_j, t_n)\Big)$ for $(i, j, n) \in [|0, N_x + 1|] \times [|0, N_y + 1|] \times [|0, N_t|]$.

We will calculate the $\Big(u(x_i, y_j, t_n),\ v(x_i, y_j, t_n)\Big)$ numerically in space domain's interior, that is to say for $(i, j) \in [|1, N_x|] \times [|1, N_y|]$, the boundary values being given by the zero-flux condition.

The error made by our numerical method will be given by Taylor expansion formulae.

3.2 Discretization of the Laplacian

To discretize the equations, and especially the expression of the Laplacian operator, we will use Taylor's expansion, assuming that u and v are continuously differentiable, at least of class $\mathcal{C}^2$ over time and at least of class $\mathcal{C}^4$ over space. Since the method is identical for both u and v, we will only detail it for u.

First, let's write Taylor's expansion between x_{i+1} and x_i, and between x_{i-1} and x_i :

$$\begin{aligned} u(x_{i+1}, y_j, t_n) &= \sum_{k=0}^{3} \frac{\Delta x^k}{k!} \frac{\partial^k u}{\partial x^k}(x_i, y_j, t_n) \; + O(\Delta x^4) \\ u(x_{i-1}, y_j, t_n) &= \sum_{k=0}^{3} \frac{(-\Delta x)^k}{k!} \frac{\partial^k u}{\partial x^k}(x_i, y_j, t_n) \; + O(\Delta x^4) \end{aligned}$$

After summing these two expressions, we obtain :

$$u(x_{i+1}, y_j, t_n) + u(x_{i-1}, y_j, t_n) = 2u(x_i, y_j, t_n) + \Delta x^2 \frac{\partial^2 u}{\partial x^2}(x_i, y_j, t_n) + O_{\Delta x \to 0}(\Delta x^4)$$

which can be re-written :

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j, t_n) = \frac{1}{\Delta x^2}\Big(u(x_{i+1}, y_j, t_n) + u(x_{i-1}, y_j, t_n) - 2u(x_i, y_j, t_n)\Big) + O(\Delta x^2)$$

Likewise, we obtain a similar formula for the y variable :

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j, t_n) = \frac{1}{\Delta y^2}\Big(u(x_i, y_{j+1}, t_n) + u(x_i, y_{j-1}, t_n) - 2u(x_i, y_j, t_n)\Big) + O(\Delta y^2)$$

Finally we get the following expression of the Laplacian operator :

$$\begin{aligned} \Delta u(x_i, y_j, t_n) &= \frac{\partial^2 u}{\partial x^2}(x_i, y_j, t_n) + \frac{\partial^2 u}{\partial y^2}(x_i, y_j, t_n) \\ &= \frac{1}{\Delta x^2}\Big(u(x_{i+1}, y_j, t_n) + u(x_{i-1}, y_j, t_n)\Big) \\ &+ \frac{1}{\Delta y^2}\Big(u(x_i, y_{j+1}, t_n) + u(x_i, y_{j-1}, t_n)\Big) \\ &- 2\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) u(x_i, y_j, t_n) \; + O(\Delta x^2 + \Delta y^2) \end{aligned}$$

3.3 Boundary Conditions

As for the boundary conditions, we will also use Taylor's expansion :

$$u(x_{N_x}, y_j, t_n) = \ u(x_{N_x+1}, y_j, t_n) - \Delta x \underbrace{\frac{\partial u}{\partial x}(x_{N_x+1}, y_j, t_n)}_{=0} + O(\Delta x^2)$$

$$u(x_1, y_j, t_n) = \ u(x_0, y_j, t_n) + \Delta x \underbrace{\frac{\partial u}{\partial x}(x_0, y_j, t_n)}_{=0} + O(\Delta x^2)$$

$$u\big(x_i, y_{N_y}, t_n\big) = \ u\big(x_i, y_{N_y+1}, t_n\big) - \Delta y \underbrace{\frac{\partial u}{\partial y}\big(x_i, y_{N_y+1}, t_n\big)}_{=0} + O(\Delta y^2)$$

$$u(x_i, y_1, t_n) = \ u(x_i, y_0, t_n) + \Delta y \underbrace{\frac{\partial u}{\partial y}(x_i, y_0, t_n)}_{=0} + O(\Delta y^2)$$

which can be summarized as :

$$\begin{cases} u(x_{N_x}, y_j, t_n) = \ u(x_{N_x+1}, y_j, t_n) + O(\Delta x^2) \\ u(x_1, y_j, t_n) = u(x_0, y_j, t_n) + O(\Delta x^2) \\ u\big(x_i, y_{N_y}, t_n\big) = \ u\big(x_i, y_{N_y+1}, t_n\big) + O(\Delta y^2) \\ u(x_i, y_1, t_n) = \ u(x_i, y_0, t_n) + O(\Delta y^2) \end{cases}$$

3.4 Time Integration

To integrate the equations over time, we will consider two options. The first one consists in writing the second order Taylor expansion of time derivative between t_{n+1} and t_n :

$$\frac{\partial u}{\partial t}(x_i, y_j, t_n) = \frac{1}{\Delta t}\Big(u(x_i, y_j, t_{n+1}) - u(x_i, y_j, t_n)\Big) + O(\Delta t^2)$$

which gives a Euler-forward based discrete system with a local truncation error on the order of $O(\Delta \mathrm{t} + \Delta x^2 + \Delta y^2)$:

$\forall\ i \in [|1, N_x|], \forall j \in \big[\big|1, N_y\big|\big], \forall t \in [|0, N_t - 1|]$

$$\begin{cases} u_{i,j}^{n+1} = \ u_{i,j}^n + \Delta t \left(\gamma f(u_{i,j}^n, v_{i,j}^n) + \dfrac{u_{i_{-1},j}^n + u_{i_{+1},j}^n}{\Delta x^2} + \dfrac{u_{i,j_{-1}}^n + u_{i,j_{+1}}^n}{\Delta y^2} - 2\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2}\right) u_{i,j}^n\right) \\ v_{i,j}^{n+1} = \ v_{i,j}^n + \Delta t \left(\gamma g(u_{i,j}^n, v_{i,j}^n) + d\left(\dfrac{v_{i_{-1},j}^n + v_{i_{+1},j}^n}{\Delta x^2} + \dfrac{v_{i,j_{-1}}^n + v_{i,j_{+1}}^n}{\Delta y^2} - 2\left(\dfrac{1}{\Delta x^2} + \dfrac{1}{\Delta y^2}\right) v_{i,j}^n\right)\right) \end{cases}$$

where :
- $i_{-1} = max(1, i-1)$
- $i_{+1} = min(i+1, N_x)$
- $j_{-1} = max(1, j-1)$
- $j_{+1} = min(j+1, N_y)$

Note that this notation ensures that the zero-flux condition is respected when i or j take border values, without modifying the original expression when they don't.

For better accuracy and in order to avoid divergence, we will consider a Runge-Kutta-4 time integration which gives a discrete system with a local truncation error on the order of $(\Delta t^5 + \Delta x^2 + \Delta y^2)$. The expression for that integration method is a quite heavy in the "scalar" case. In the next section, we will provide matrix-wise expressions for that method, and also for the one above.

4 Numerical Model

To solve these equations, we will use MATLAB. That language is matrix-oriented and requires avoiding loops as much as possible to decrease CPU time. To do so, we will define vector-wise expressions of our unknowns $(u_{i,j}^n, v_{i,j}^n)_{(i,j,n)\in[|1,N_x|]\times[|1,N_y|]\times[|1,N_t|]}$.

For $n \in [|1, N_t|]$, let :

$$U^n = \begin{pmatrix} U_{\bullet 1}^n \\ U_{\bullet 2}^n \\ \vdots \\ U_{\bullet N_y}^n \end{pmatrix} \in \mathcal{M}_{N_x N_y,1}(\mathbb{R}) \qquad V^n = \begin{pmatrix} V_{\bullet 1}^n \\ V_{\bullet 2}^n \\ \vdots \\ V_{\bullet N_y}^n \end{pmatrix} \in \mathcal{M}_{N_x N_y,1}(\mathbb{R})$$

where

$$\forall j \in [|1, N_y|], \qquad U_{\bullet j}^n = \begin{pmatrix} u_{1,j}^n \\ u_{2,j}^n \\ \vdots \\ u_{N_x,j}^n \end{pmatrix} \in \mathcal{M}_{N_x,1}(\mathbb{R}) \qquad V_{\bullet j}^n = \begin{pmatrix} v_{1,j}^n \\ v_{2,j}^n \\ \vdots \\ v_{N_x,j}^n \end{pmatrix} \in \mathcal{M}_{N_x,1}(\mathbb{R})$$

Now, we can rewrite the expression of the Laplacian operator taking into account the zero-flux boundary conditions :

$$\Delta U^n = \begin{pmatrix} \Delta U_{\bullet 1}^n \\ \Delta U_{\bullet 2}^n \\ \vdots \\ \vdots \\ \Delta U_{\bullet N_y}^n \end{pmatrix} = AU^n \in \mathcal{M}_{N_x N_y,1}(\mathbb{R})$$

with

$$A = \begin{pmatrix} B_1 & I & & & & \\ I & B_0 & I & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & I & B_0 & I \\ & & & & I & B_1 \end{pmatrix} \in \mathcal{M}_{N_x N_y}(\mathbb{R}) \qquad \Delta U_{\bullet j}^n = \begin{pmatrix} \Delta u_{1,j}^n \\ \Delta u_{2,j}^n \\ \vdots \\ \Delta_{u_{N_x,j}^n} \end{pmatrix} \in \mathcal{M}_{N_x,1}(\mathbb{R})$$

$$B_0 = \begin{pmatrix} (\frac{-1}{\Delta x^2}+\frac{-2}{\Delta y^2}) & \frac{1}{\Delta x^2} & & & \\ \frac{1}{\Delta x^2} & (\frac{-2}{\Delta x^2}+\frac{-2}{\Delta y^2}) & \frac{1}{\Delta x^2} & & \\ \ddots & \ddots & \ddots & & \\ & \frac{1}{\Delta x^2} & (\frac{-2}{\Delta x^2}+\frac{-2}{\Delta y^2}) & \frac{1}{\Delta x^2} & \\ & & \frac{1}{\Delta x^2} & (\frac{-1}{\Delta x^2}+\frac{-2}{\Delta y^2}) \end{pmatrix} \in \mathcal{M}_{N_x}(\mathbb{R})$$

$$B_1 = \begin{pmatrix} (\frac{-1}{\Delta x^2}+\frac{-1}{\Delta y^2}) & \frac{1}{\Delta x^2} & & & \\ \frac{1}{\Delta x^2} & (\frac{-2}{\Delta x^2}+\frac{-1}{\Delta y^2}) & \frac{1}{\Delta x^2} & & \\ \ddots & \ddots & \ddots & & \\ & \frac{1}{\Delta x^2} & (\frac{-2}{\Delta x^2}+\frac{-1}{\Delta y^2}) & \frac{1}{\Delta x^2} & \\ & & \frac{1}{\Delta x^2} & (\frac{-1}{\Delta x^2}+\frac{-1}{\Delta y^2}) \end{pmatrix} \in \mathcal{M}_{N_x}(\mathbb{R})$$

$$I = \begin{pmatrix} \frac{1}{\Delta y^2} & & \\ & \ddots & \\ & & \frac{1}{\Delta y^2} \end{pmatrix} \in \mathcal{M}_{N_x}(\mathbb{R})$$

Thanks to that matrix expression of the Laplacian operator, we can re-write the time integration methods :

Euler Forward

$$\begin{cases} U^{n+1} = U^n + \Delta t(\gamma F(U^n, V^n) + AU^n) \\ V^{n+1} = U^n + \Delta t(\gamma G(U^n, V^n) + dAU^n) \end{cases}$$

where

$$F(U^n, V^n) = \begin{pmatrix} F(U^n_{\bullet 1}, V^n_{\bullet 1}) \\ F(U^n_{\bullet 2}, V^n_{\bullet 2}) \\ \vdots \\ F\big(U^n_{\bullet N_y}, V^n_{\bullet N_y}\big) \end{pmatrix} \mathcal{M}_{N_x N_y, 1}(\mathbb{R}) \qquad G(U^n, V^n) = \begin{pmatrix} G(U^n_{\bullet 1}, V^n_{\bullet 1}) \\ G(U^n_{\bullet 2}, V^n_{\bullet 2}) \\ \vdots \\ G(U^n_{\bullet N_y}, V^n_{\bullet N_y}) \end{pmatrix} \mathcal{M}_{N_x N_y, 1}(\mathbb{R})$$

$$F(U^n_{\bullet j}, V^n_{\bullet j}) = \begin{pmatrix} f(U^n_{1,j}, V^n_{1,j}) \\ f(U^n_{2,j}, V^n_{2,j}) \\ \vdots \\ f\big(U^n_{N_x,j}, V^n_{N_x,j}\big) \end{pmatrix} \mathcal{M}_{N_x,1}(\mathbb{R}) \qquad G(U^n_{\bullet j}, V^n_{\bullet j}) = \begin{pmatrix} g(U^n_{1,j}, V^n_{1,j}) \\ g(U^n_{2,j}, V^n_{2,j}) \\ \vdots \\ g\big(U^n_{N_x,j}, V^n_{N_x,j}\big) \end{pmatrix} \mathcal{M}_{N_x,1}(\mathbb{R})$$

Runge-Kutta 4

$$\begin{cases} (k_1, p_1) = f_{RK4}(U^n, V^n) \\ (k_2, p_2) = f_{RK4}\big(U^n + \frac{\Delta t}{2} k_1, V^n + \frac{\Delta t}{2} p_1\big) \\ (k_3, p_3) = f_{RK4}\big(U^n + \frac{\Delta t}{2} k_2, V^n + \frac{\Delta t}{2} p_2\big) \\ (k_4, p_4) = f_{RK4}(U^n + \Delta t k_3, V^n + \Delta t p_3) \\ U^{n+1} = U^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ V^{n+1} = V^n + \frac{\Delta t}{6}(p_1 + 2p_2 + 2p_3 + p_4) \end{cases}$$

where $\quad f_{RK4}(k, p) = (\gamma f(k, p) + Ak \ , \ \gamma g(k, p) + dAp)$

For all the simulations presented in the next section, we are going to use Runge-Kutta 4 method.

5 Results and Discussions

In this section, we will provide some MATLAB simulations of the above mentioned numerical model. Let us remind ourselves that the goal of this report is to show that the equations in question can explain pattern formation in animal coats. Conditions for pattern formation can be found in literature, in Murray's book *Mathematical Biology* [3] for instance, but they won't be addressed in that report. Instead, as a guide towards the right conditions to obtain patterns, we will use values provided by both Murray's book [3] and Vidiani's paper [1].

5.1 Initial Conditions

As suggested by the above mentioned references, we will choose initial conditions as follows :

$$\forall (i,j) \in [|1, N_x|] \times [|1, N_y|], \qquad \begin{cases} u^0_{i,j} = u_s + \varepsilon_{i,j} \\ v^0_{i,j} = v_s + \theta_{i,j} \end{cases}$$

where
- $(u_s, v_s) \in \mathbb{R}^2$ is the "diffusionless steady state" defined by $\begin{cases} f(u_s, v_s) = 0 \\ g(u_s, v_s) = 0 \end{cases}$
- $(\varepsilon_{i,j}, \theta_{i,j}) \in \mathbb{R}^2$ are random values between taken within $[-\delta, \delta]$, with $\delta > 0$

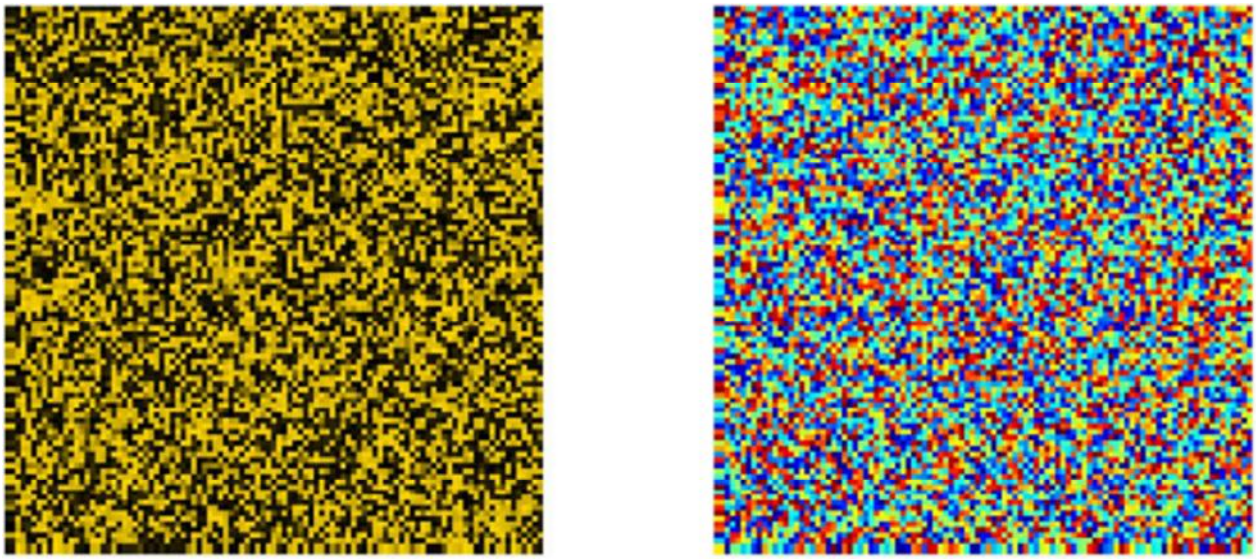
5.2 Cheetah

Here's a first set (S_0) of parameters we will use to obtain patterns :

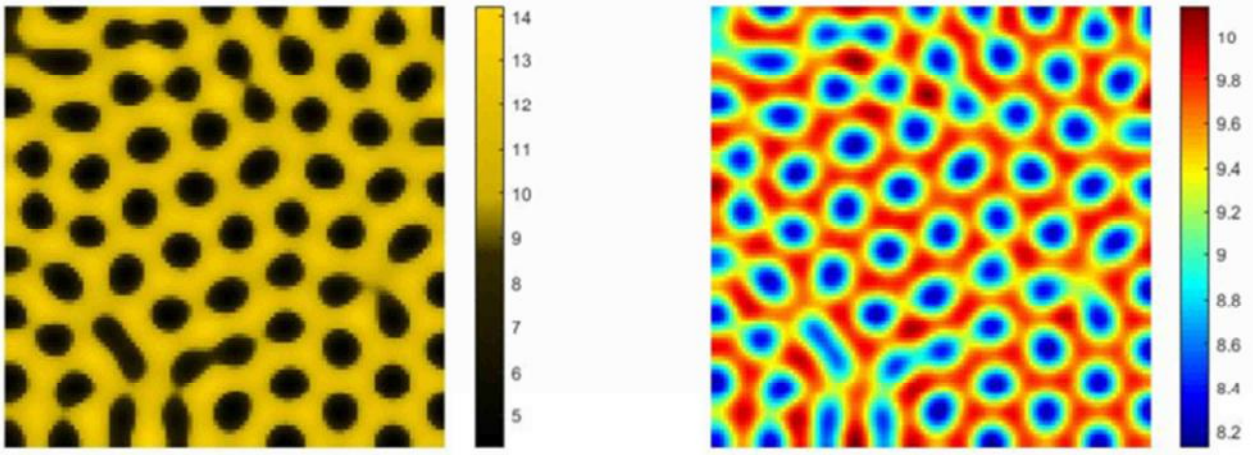
- $(x_{min}, x_{max}) = (-10,10)$
- $(y_{min}, y_{max}) = (-10,10)$
- $d = 10$
- $\gamma = 9$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.1$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Note that T is chosen so that the steady state can be established.

The initial conditions are represented in the image below. The concentration of activator u is represented on the left, and the concentration of inhibitor is represented on the right.



Here is the result of the simulation for the (S_0) set of parameters. The activator u (on the left) and the inhibitor v (on the right) show the same pattern, but are colored with different color maps. The color map of u is "centered" about u_s, i.e. black below $u_s = 9$ and gold above.



Results for set (S_0) ; CPU time : $t_{CPU} = 170.76\ s$

The pattern obtained at the steady state is composed of evenly spaced dots. Note that this pattern that is similar to the one we can find on cheetah fur, as illustrated opposite.

5.2 Parameter Dependency

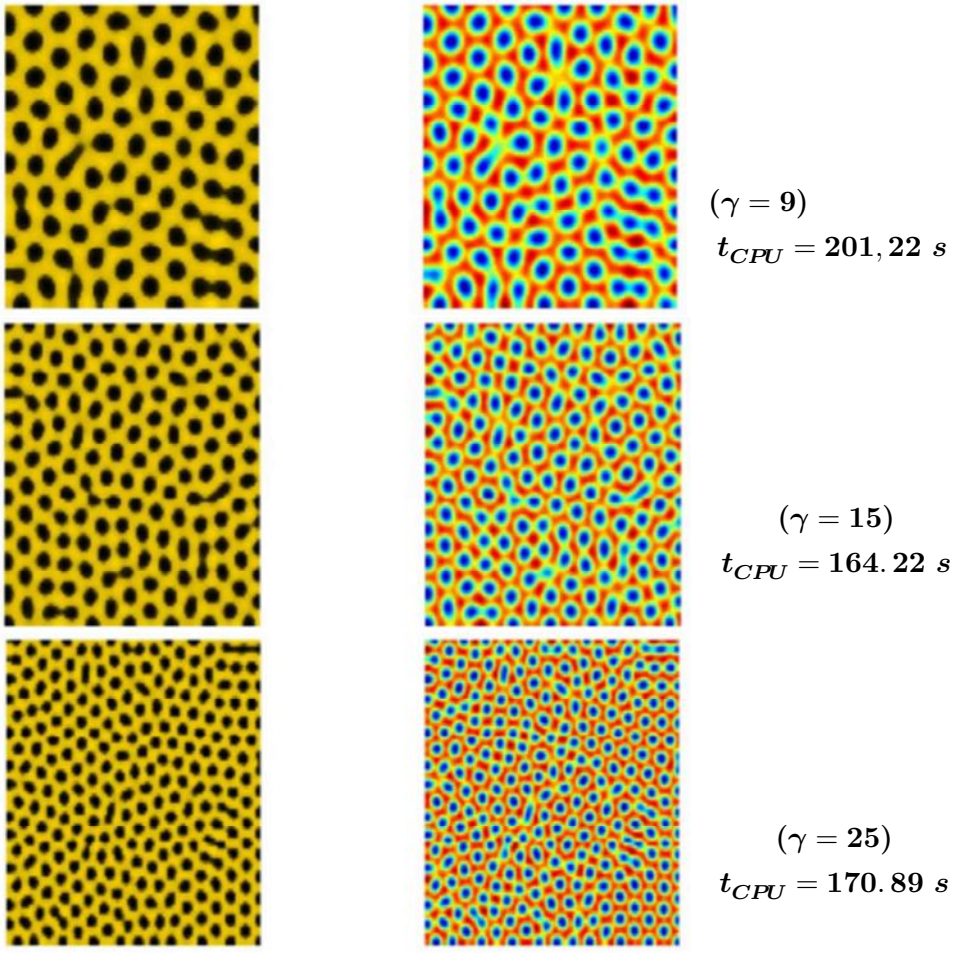
In this section, we will focus on the influence of some parameters of the system (within stability conditions) and see what role they have on its evolution.

Influence of γ

All other parameters remaining unchanged from set (S_γ), we will try several values of for γ and see what happens. Here is the detail of parameter set (S_γ) :

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = 10$
- $\gamma = \{9,15,25\}$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.1$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Here are the results for set (S_γ) :



$(\boldsymbol{\gamma = 9})$
$\boldsymbol{t_{CPU} = 201,22\ s}$

$(\boldsymbol{\gamma = 15})$
$\boldsymbol{t_{CPU} = 164.22\ s}$

$(\boldsymbol{\gamma = 25})$
$\boldsymbol{t_{CPU} = 170.89\ s}$

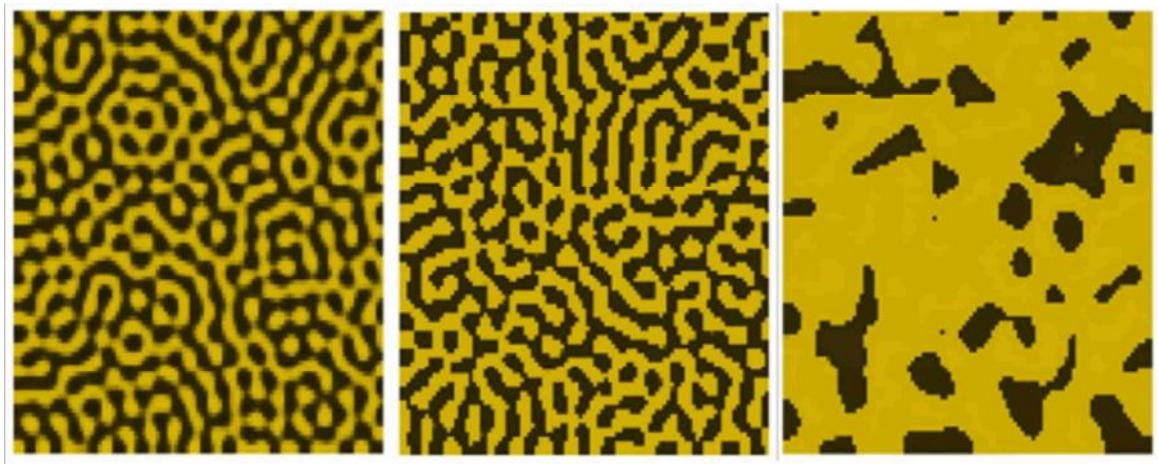
Note that the width of the dots and the space between them decreases as γ increases. As a matter of fact, γ is the coefficient of the reaction term, which means that the relative importance reaction term, in relation to diffusion, increases as γ increases. This parameter could be interpreted as a spatial scale for the domain or as a way to choose a spatial mode for the patterns.

Influence of d

All other parameters remaining unchanged from set (S_d), we will try several values of for d and see what happens. Here is the detail of parameter set (S_d) :

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = \{3,6,8\}$
- $\gamma = 25$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.1$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Here are the results obtained for set (S_d) :



$\boldsymbol{d = 8}$; $\boldsymbol{t_{CPU} = 165.45\ s}$ $\boldsymbol{d = 6}$; $\boldsymbol{t_{CPU} = 163.70\ s}$ $\boldsymbol{d = 3}$; $\boldsymbol{t_{CPU} = 167.82\ s}$

We can observe that when d decreases, which means that the ratio of inhibitor diffusion over activator diffusion decreases, the previously obtained dots get linked, progressively become lines and finally lose periodicity when the ratio is not sufficiently big anymore ($d = \frac{D_v}{D_u} = 3$ in that case).

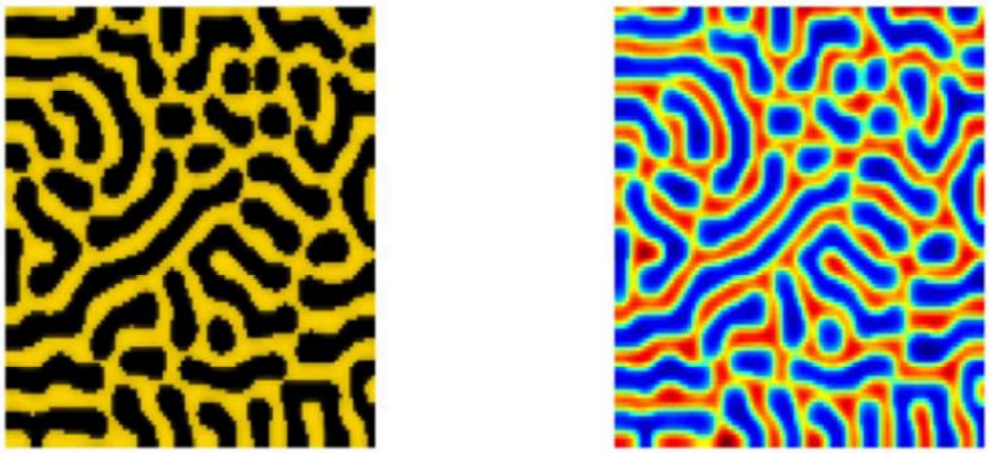
5.3 King Cheetah

Starting from a set of type (S_γ), if we slightly change some parameters, we can eventually get some interesting patterns that resemble the ones from sections 5.2 and 5.1, but that are quite different.

Here's the set (S_{KC}) we're going to use in that case :

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = 13$
- $\gamma = 14.5$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.15$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Here are the results we get for (S_{KC}) :



Results for set (S_{KC}) ; CPU time : $t_{CPU} = 162.04\ s$

This pattern turns out to be similar to the one of found on King Cheetah fur. It is made up of lines, linked dots and uneven misshapen dots, just like the one of King Cheetah, as illustrated below.

The fact that a slight change in parameters from the ones that give a Cheetah-like pattern eventually give the one of king cheetah is very interesting. It might be interpreted as a slight change in the gene pool that creates a different outline for the spatial distribution of morphogens, but still comparable to the "original" one.

Note that if we change the color map, it will somehow resemble the patterns found on some parts of tapirs.

5.4 Lynx

With another set of values, call it (S_L) we can get patterns found on a lynx.

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = 10.9$
- $\gamma = 17$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.2$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.45$
- $\alpha = 1.5$

Results for set (S_{KC}) ; CPU time : $t_{CPU} = 194.47\ s$

5.5 Other Patterns

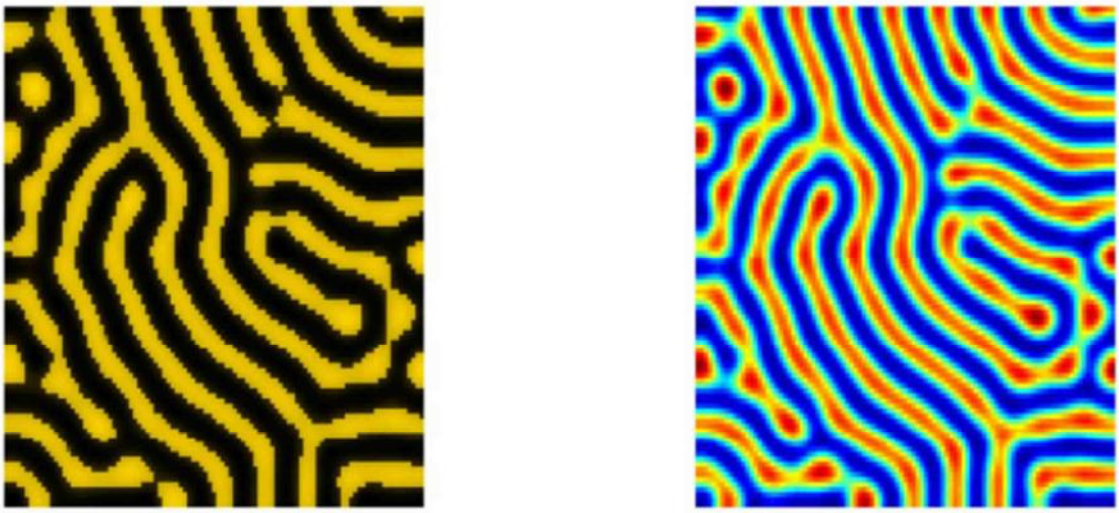
This system of equations can also lead to other patterns. For example, here are two interesting patterns (even though not associated to any animal coat) with their respective set of parameters.

First pattern

Set of parameters (S_1)

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = 12$
- $\gamma = 9$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.1$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Result



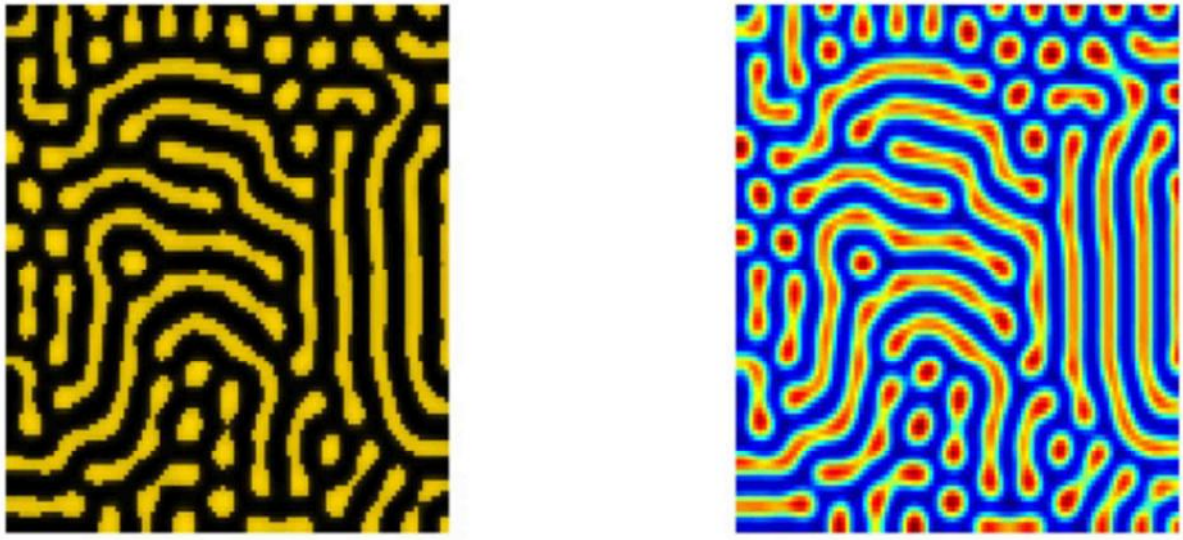
Results for set (S_1) ; CPU time : $t_{CPU} = 166.47\ s$

Second pattern

Set of parameters (S_2)

- $(x_{min}, x_{max}) = (-12,12)$
- $(y_{min}, y_{max}) = (-12,12)$
- $d = 14$
- $\gamma = 16$
- $N_x = 100$
- $N_y = 100$
- $\delta = 4$
- $u_s = 10$
- $T = 20$
- $N_t = 20000$
- $K = 0.1$
- $u_v = 9$
- $a = 92$
- $b = 64$
- $\rho = 18.5$
- $\alpha = 1.5$

Result



Results for set (S_2) ; CPU time : $t_{CPU} = 171.25\ s$

5.6 Skin Wound

In that section, we will try to simulate the effect of a wound on a growing skin. To do so, we will build initial conditions by multiplying the center values of the steady state obtained from set ($S_{\gamma=9}$) by random values contained within [0,1.5]. The initial conditions are represented below.

Then we run a simulation by using the same set of parameters. The result is represented below.

The situation that is modeled here is a growing skin that happens to be perturbed during its formation process. We can see that the result is close to the unwounded case, whether it is inside or outside the wound. Therefore, judging from this model, we can conclude that fur patterns are strongly determined by gene pools, independently of environmental perturbations.

6 Conclusion

From a mathematical reaction-diffusion model designed from physicochemical considerations, numerical computations show that these equations are quite reliable to explain natural phenomena such as animal fur patterns and skin wound healing in relation with genetics and biochemistry.

Besides patterns from other animals that can be explained with reaction-diffusion equations, such systems can also model miscellaneous natural phenomena from epidemic spreads, to plant structures, embryonic and organ development, as well as spatial distribution of stars and gas in galaxies

In other words, reaction-diffusion equations seem to have a lot of potential and are worth studying both numerically and analytically.

Appendix

Computer Specifications

The computer that has been used for all the above mentioned simulations is an ASUS R752L with a Windows 10 operating system. Here are its specifications :

- Processor : Intel Core i7-4510U CPU @ 2.00GHz 2.60GHz (2 cores)
- RAM : 6.00 Go
- Graphic Card : NVIDIA GEFORCE GTX 850M

Program List

Here is the list of the mains scripts that have been used to compute and plot the different solutions mentioned above.

MainReacDiff

```
%% --------------- Clear Workspace and close all figures ------------------
close all
clear


%% --------------- Discretization of Time & 2D space ----------------------

% Discretization of time domain
T = 20 ;
Nt = 20000 ;
animStep = 40 ;

% Discretization of x domain
xmin = -12 ;
xmax = 12 ;
Nx = 100 ;
Tx = abs(xmax-xmin) ;
wx = 2*pi/Tx ;

% Discretization of y domain
ymin = -12 ;
ymax = 12 ;
Ny = 100 ;
Ty = abs(ymax-ymin) ;
wy = 2*pi/Ty ;


%% ---------------------------- Systems ----------------------------------

% Diffusion coefficients
d = 10 ;
gamma = 9 ;


%---------------------------- Schnakenberg --------------------------------
% % Parameters
% a = 0.2 ;
% b = 0.6 ;
% % Reaction Terms
% F = @(u,v) a - u + (u.^2).*v ;
% G = @(u,v) b - (u.^2).*v ;
% % Initial conditions
% u_in = @(x,y) 0.5*sin( 2*pi .* repmat(x,1,Ny) ).*sin( 2*pi .* epmat(y,Nx,1) ) ;
% v_in = @(x,y) 0.5*sin( 3*pi .* repmat(y,Nx,1) ).*sin( 3*pi .*
repmat(x,1,Ny) ) ;
```

```
%------------------------ Substrate Inhibition --------------------------
% Parameters
a = 92 ;
b = 64 ;
K = 0.1 ;
alpha = 1.5 ;
rho = 18.5 ;
% Reaction Terms
F = @(u,v) a - u - rho*u.*v ./ (1 + u + K*u.^2) ;
G = @(u,v) alpha*(b-v) - rho*u.*v ./ (1 + u + K*u.^2) ;
% Initial conditions
% u_in = @(x,y) 1 + sin( wx*pi * repmat(x,1,Ny) ).*sin( wy*pi *
repmat(y,Nx,1) ) ;
% v_in = @(x,y) 1  + sin( 3*wx*pi * repmat(y,Nx,1) ).*sin( 3*wy*pi *
repmat(x,1,Ny) ) ;
u_in = @(x,y) 10 + 4* 2*(rand(Nx,Ny)-0.5) ;
v_in = @(x,y)  9 + 4* 2*(rand(Nx,Ny)-0.5) ;



%% --------------------------- Resolution --------------------------------
t1 = cputime ;
[xf,yf,tf,MUf,MVf] = FiniteDiffRK4( xmin, xmax, Nx, ymin, ymax, Ny , T, Nt,...
u_in, v_in, d, gamma, F, G) ;
tcpu = cputime-t1 ;


%% --------------------------- Animation ---------------------------------
% Animate2(xf,yf,tf,MUf,MVf,animStep)


%% ----------------------------- Save ------------------------------------
save('SubInhib_n.mat','xmin','xmax','Nx','ymin','ymax','Ny','T','Nt',...
     'a','b','d','gamma','K','alpha','rho','u_in','v_in','F','G',...
     'animStep','tcpu') ;
```

MainWound

```
%% --------------- Clear Workspace and close all figures ------------------
close all
clear


%% ----------------------------- Load ---------------------------------------
load SubInhib_Cheetah_P1
load Wound_Cheetah_P1


%% -------------------------- Artificial Wound ------------------------------
Uin = U_cheetah(2:Nx+1,2:Ny+1) ;
Uin(25:75,25:75) = Uin(25:75,25:75) .* (1.5*rand(51,51));
Vin = V_cheetah(2:Nx+1,2:Ny+1) ;
Vin(25:75,25:75) = Vin(25:75,25:75) .* (1.5*rand(51,51));


%% -------------------------- Resolution ------------------------------------
t1 = cputime ;
[xf,yf,tf,MUf,MVf] = FiniteDiffRK4Wound( xmin, xmax, Nx, ymin, ymax, Ny , T, Nt,
Uin, Vin, d, gamma, F, G) ;
tcpu = cputime-t1 ;


%% -------------------------- Animation -------------------------------------
%Animate3(xf,yf,tf,MUf,MVf,animStep)


%% ----------------------------- Save ---------------------------------------
% save('SubInhib_n.mat','xmin','xmax','Nx','ymin','ymax','Ny','T','Nt',...
%      'a','b','d','gamma','K','alpha','rho','u_in','v_in','F','G',...
%      'animStep','tcpu') ;
```

FiniteDiffRK4

```
function [ xf, yf, tf, MUf, MVf  ] = FiniteDiffRK4( xmin, xmax, Nx, ymin, ymax,
Ny , T, Nt, u_in, v_in, d, gamma, F, G)

%% Discretization of space time domain

% x discretization (column)
dx = (xmax-xmin)/(Nx+1);
x = (xmin+dx:dx:xmax-dx)';

% y discretization (row)
dy = (ymax-ymin)/(Ny+1);
y = (ymin+dy:dy:ymax-dy);

% t discretization
dt = T/Nt;
tf = 0:dt:T;




%% Matrice A
A = MatriceA_ZeroFlux(Nx, dx, Ny, dy) ;


%% Time derivative formula for RK4
Fsim = @(U,V) gamma*F(U,V) +   A*U ;
Gsim = @(U,V) gamma*G(U,V) + d*A*V ;



%% Resolution

% Storage
MUf = zeros(Nx+2, Ny+2, Nt+1) ;
MVf = zeros(Nx+2, Ny+2, Nt+1) ;

% Initialization
    % u function
U0 = u_in(x,y) ;
MUf(2:Nx+1, 2:Ny+1, 1 ) = U0 ;
U0 = reshape(U0,Nx*Ny,1) ;
    % u function
V0 = v_in(x,y) ;
MVf(2:Nx+1, 2:Ny+1, 1 ) = V0 ;
V0 = reshape(V0,Nx*Ny,1) ;

% Iterations
for n = 1:Nt
    % Calculation by RK4
    k1 = Fsim(U0,V0) ;                    p1 = Gsim(U0,V0) ;
    k2 = Fsim(U0+dt/2*k1, V0+dt/2*p1) ; p2 = Gsim(U0+dt/2*k1, V0+dt/2*p1) ;
    k3 = Fsim(U0+dt/2*k2, V0+dt/2*p2) ; p3 = Gsim(U0+dt/2*k2, V0+dt/2*p2) ;
    k4 = Fsim(U0+dt*k3, V0+dt*p3) ;      p4 = Gsim(U0+dt*k3, V0+dt*p3) ;
    U1 = U0 + dt/6 * (k1 + 2*k2 + 2*k3 + k4) ;
```

```
    V1 = V0 + dt/6 * (p1 + 2*p2 + 2*p3 + p4) ;
    % Storage
    MUf(2:Nx+1, 2:Ny+1, n+1 ) = reshape(U1, Nx, Ny) ;
    MVf(2:Nx+1, 2:Ny+1, n+1 ) = reshape(V1, Nx, Ny) ;
    % Updating state vectors
    U0 = U1 ;
    V0 = V1 ;
    % Progress
    ['Calculation ongoing : ',num2str(n/Nt*100),' %']
end


%% Storage - Boundaries of the Space Domain (for graphic rendering)
% Left
MUf(1, 2:Ny+1, :) = MUf(2, 2:Ny+1, :) ;
MVf(1, 2:Ny+1, :) = MVf(2, 2:Ny+1, :) ;
% Right
MUf(Nx+2, 2:Ny+1, :) = MUf(Nx+1, 2:Ny+1, :) ;
MVf(Nx+2, 2:Ny+1, :) = MVf(Nx+1, 2:Ny+1, :) ;
% Bottom
MUf(2:Nx+1, 1, :) = MUf(2:Nx+1, 2, :) ;
MVf(2:Nx+1, 1, :) = MVf(2:Nx+1, 2, :) ;
% Top
MUf(2:Nx+1, Ny+2, :) = MUf(2:Nx+1, Ny+1, :) ;
MVf(2:Nx+1, Ny+2, :) = MVf(2:Nx+1, Ny+1, :) ;
% Corners
MUf(1, 1, :) = 1/3*( MUf(1, 2, :) + MUf(2, 1, :) + MUf(2, 2, :) ) ;
MVf(1, 1, :) = 1/3*( MVf(1, 2, :) + MVf(2, 1, :) + MVf(2, 2, :) ) ;
MUf(Nx+2, Ny+2, :) = 1/3*( MUf(Nx+1, Ny+2, :) + MUf(Nx+2, Ny+1, :) + MUf(Nx+1,
Ny+1, :) ) ;
MVf(Nx+2, Ny+2, :) = 1/3*( MVf(Nx+1, Ny+2, :) + MVf(Nx+2, Ny+1, :) + MVf(Nx+1,
Ny+1, :) ) ;
MUf(1, Ny+2, :) = 1/3*( MUf(1, Ny+1, :) + MUf(2, Ny+2, :) + MUf(2, Ny+1, :) ) ;
MVf(1, Ny+2, :) = 1/3*( MVf(1, Ny+1, :) + MVf(2, Ny+2, :) + MVf(2, Ny+1, :) ) ;
MUf(Nx+2, 1, :) = 1/3*( MUf(Nx+2, 2, :) + MUf(Nx+1, 1, :) + MUf(Nx+1, 2, :) ) ;
MVf(Nx+2, 1, :) = 1/3*( MVf(Nx+2, 2, :) + MVf(Nx+1, 1, :) + MVf(Nx+1, 2, :) ) ;


%% Complete resolution data
    % x vector
xf = zeros(Nx+2,1);
xf(2:Nx+1) = x;
xf(1) = xmin;
xf(Nx+2) = xmax;
    % y vector
yf = zeros(Ny+2,1);
yf(2:Ny+1) = y;
yf(1) = ymin;
yf(Ny+2) = ymax;



end
```

FiniteDiffRK4Wound

```
function [ xf, yf, tf, MUf, MVf  ] = FiniteDiffRK4Wound( xmin, xmax, Nx, ymin,
ymax, Ny , T, Nt, Uin, Vin, d, gamma, F, G)

%% Discretization of space time domain

% x discretization (column)
dx = (xmax-xmin)/(Nx+1);
x = (xmin+dx:dx:xmax-dx)';

% y discretization (row)
dy = (ymax-ymin)/(Ny+1);
y = (ymin+dy:dy:ymax-dy);

% t discretization
dt = T/Nt;
tf = 0:dt:T;




%% Matrice A
A = MatriceA_ZeroFlux(Nx, dx, Ny, dy) ;


%% Time derivative formula for RK4
Fsim = @(U,V) gamma*F(U,V) +   A*U ;
Gsim = @(U,V) gamma*G(U,V) + d*A*V ;



%% Resolution

% Storage
MUf = zeros(Nx+2, Ny+2, Nt+1) ;
MVf = zeros(Nx+2, Ny+2, Nt+1) ;

% Initialization
    % u function
U0 = Uin ;
MUf(2:Nx+1, 2:Ny+1, 1 ) = U0 ;
U0 = reshape(U0,Nx*Ny,1) ;
    % u function
V0 = Vin ;
MVf(2:Nx+1, 2:Ny+1, 1 ) = V0 ;
V0 = reshape(V0,Nx*Ny,1) ;

% Iterations
for n = 1:Nt
    % Calculation by RK4
    k1 = Fsim(U0,V0) ;                    p1 = Gsim(U0,V0) ;
    k2 = Fsim(U0+dt/2*k1, V0+dt/2*p1) ; p2 = Gsim(U0+dt/2*k1, V0+dt/2*p1) ;
    k3 = Fsim(U0+dt/2*k2, V0+dt/2*p2) ; p3 = Gsim(U0+dt/2*k2, V0+dt/2*p2) ;
    k4 = Fsim(U0+dt*k3, V0+dt*p3) ;      p4 = Gsim(U0+dt*k3, V0+dt*p3) ;
    U1 = U0 + dt/6 * (k1 + 2*k2 + 2*k3 + k4) ;
    V1 = V0 + dt/6 * (p1 + 2*p2 + 2*p3 + p4) ;
```

```
    % Storage
    MUf(2:Nx+1, 2:Ny+1, n+1 ) = reshape(U1, Nx, Ny) ;
    MVf(2:Nx+1, 2:Ny+1, n+1 ) = reshape(V1, Nx, Ny) ;
    % Updating state vectors
    U0 = U1 ;
    V0 = V1 ;
    % Progress
    ['Calculation ongoing : ',num2str(n/Nt*100),' %']
end


%% Storage - Boundaries of the Space Domain (for graphic rendering)
% Left
MUf(1, 2:Ny+1, :) = MUf(2, 2:Ny+1, :) ;
MVf(1, 2:Ny+1, :) = MVf(2, 2:Ny+1, :) ;
% Right
MUf(Nx+2, 2:Ny+1, :) = MUf(Nx+1, 2:Ny+1, :) ;
MVf(Nx+2, 2:Ny+1, :) = MVf(Nx+1, 2:Ny+1, :) ;
% Bottom
MUf(2:Nx+1, 1, :) = MUf(2:Nx+1, 2, :) ;
MVf(2:Nx+1, 1, :) = MVf(2:Nx+1, 2, :) ;
% Top
MUf(2:Nx+1, Ny+2, :) = MUf(2:Nx+1, Ny+1, :) ;
MVf(2:Nx+1, Ny+2, :) = MVf(2:Nx+1, Ny+1, :) ;
% Corners
MUf(1, 1, :) = 1/3*( MUf(1, 2, :) + MUf(2, 1, :) + MUf(2, 2, :) ) ;
MVf(1, 1, :) = 1/3*( MVf(1, 2, :) + MVf(2, 1, :) + MVf(2, 2, :) ) ;
MUf(Nx+2, Ny+2, :) = 1/3*( MUf(Nx+1, Ny+2, :) + MUf(Nx+2, Ny+1, :) + MUf(Nx+1,
Ny+1, :) ) ;
MVf(Nx+2, Ny+2, :) = 1/3*( MVf(Nx+1, Ny+2, :) + MVf(Nx+2, Ny+1, :) + MVf(Nx+1,
Ny+1, :) ) ;
MUf(1, Ny+2, :) = 1/3*( MUf(1, Ny+1, :) + MUf(2, Ny+2, :) + MUf(2, Ny+1, :) ) ;
MVf(1, Ny+2, :) = 1/3*( MVf(1, Ny+1, :) + MVf(2, Ny+2, :) + MVf(2, Ny+1, :) ) ;
MUf(Nx+2, 1, :) = 1/3*( MUf(Nx+2, 2, :) + MUf(Nx+1, 1, :) + MUf(Nx+1, 2, :) ) ;
MVf(Nx+2, 1, :) = 1/3*( MVf(Nx+2, 2, :) + MVf(Nx+1, 1, :) + MVf(Nx+1, 2, :) ) ;


%% Complete resolution data
    % x vector
xf = zeros(Nx+2,1);
xf(2:Nx+1) = x;
xf(1) = xmin;
xf(Nx+2) = xmax;
    % y vector
yf = zeros(Ny+2,1);
yf(2:Ny+1) = y;
yf(1) = ymin;
yf(Ny+2) = ymax;



end
```

Laplacian Matrix with Zero-Flux Boundary Conditions

```
function A = MatriceA_ZeroFlux(Nx,dx,Ny,dy)
% Returns the Nx*Ny matrix of 2D (plus) Laplacien operator in the case of
% zeros flux boundary condtions

% Block matrices
I = 1/dy^2 * speye(Nx,Nx) ;
D = 1/dx^2 * ( sparse(1:Nx-1,2:Nx,1,Nx,Nx) + sparse(2:Nx,1:Nx-1,1,Nx,Nx) ) ;
C1 = sparse( diag( [-1/dx^2-1/dy^2, (-2/dx^2-1/dy^2)*ones(1,Nx-2), -1/dx^2-
1/dy^2] ) ) ;
C0 = sparse( diag( [-1/dx^2-2/dy^2, (-2/dx^2-2/dy^2)*ones(1,Nx-2), -1/dx^2-
2/dy^2] ) ) ;
B1 = C1 + D ;
B0 = C0 + D ;

% Initialize Laplatian zero flux matrix
A=sparse(Nx*Ny,Nx*Ny) ;

% Fill in the matrix
A(1:Nx,1:Nx) = B1 ;
A(1:Nx,Nx+(1:Nx)) = I ;

for j=2:Ny-1
   A((j-1)*Nx+(1:Nx),(j-2)*Nx+(1:Nx)) = I ;
   A((j-1)*Nx+(1:Nx),(j-1)*Nx+(1:Nx)) = B0 ;
   A((j-1)*Nx+(1:Nx),j*Nx+(1:Nx)) = I ;
end

A((Ny-1)*Nx+(1:Nx),(Ny-1)*Nx+(1:Nx)) = B1 ;
A((Ny-1)*Nx+(1:Nx),(Ny-2)*Nx+(1:Nx)) = I ;

A = sparse(A);

end
```

Animation

```
function [] = Animate3(xf,yf,tf,MUf,MVf,animStep)

lambda = [(0:0.0001:0.2)' ; (0.2:0.1:0.8)' ; (0.8:0.0001:1)'] ;

C0 = [0,0,0] ;
C1 = [1,215/255,0] ;
Cmap = lambda*C1 + (1-lambda)*C0 ;
Cmap = [zeros(2,3) ; Cmap] ;


for i = 1:1%animStep:length(tf)
    Displot3(xf,yf,tf,MUf(:,:,i),MVf(:,:,i),i,Cmap)
    pause(0.05)
end

end
```

Plot

```
function [] = Displot3(xf,yf,tf,Uf,Vf,i,Cmap)

Nt = length(tf) ;

s1 = subplot(1,2,1) ;
surf(xf, yf, Uf', 'EdgeColor', 'None')
colormap(s1, Cmap)
% title(strcat('Temps = ',num2str((i-1)/Nt*100),' %'))
xlabel('x')
ylabel('y')
zlabel('u(x,y,t)')
view(0,90)
axis off

s2 = subplot(1,2,2);
surf(xf, yf, Vf','EdgeColor', 'None')
colormap(s2, jet)
% title(strcat('Temps = ',num2str((i-1)/Nt*100),' %'))
strcat('Temps = ',num2str((i-1)/Nt*100),' %')
xlabel('x')
ylabel('y')
zlabel('v(x,y,t)')
view(0,90)
axis off

end
```

References

[1] L.G. Vidani, Les motifs des pelages d'animaux, (2004). http://culturemath.ens.fr/maths/pdf/analyse/vidiani_motifs_pelage_animaux.pdf (accessed July 30, 2018).

[2] J.D. Murray, Mathematical Biology : An Introduction, 2002. http://www.ift.unesp.br/users/mmenezes/mathbio.pdf (accessed July 30, 2018).

[3] J.D. Murray, Mathematical Biology : Spatial Models and Biomedical Applications, 2003. http://pcleon.if.ufrgs.br/pub/listas-sistdin/MurrayII.pdf (accessed July 30, 2018).

[4] C. Kuttler, Reaction-Diffusion equations with applications, (2011). http://www-m6.ma.tum.de/~kuttler/script_reaktdiff.pdf (accessed July 30, 2018).

[5] A.M. Turing, The Chemical Basis of Morphogenesis, Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences. 237 (1952) 37–72. http://www.dna.caltech.edu/courses/cs191/paperscs191/turing.pdf (accessed July 30, 2018).