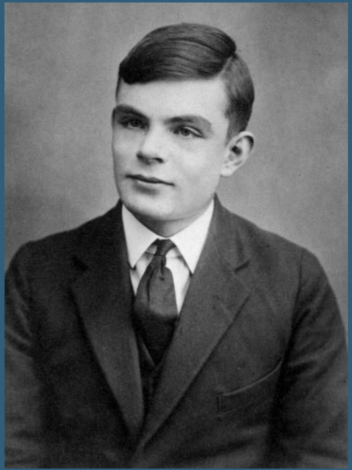
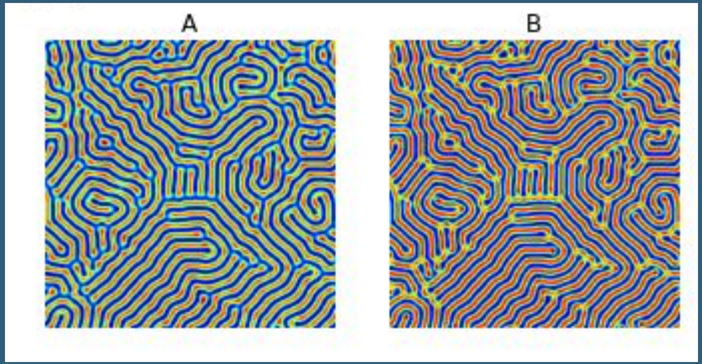
Morphogenèse et reconnaissance d’images

Histoire et explication de la morphogenèse



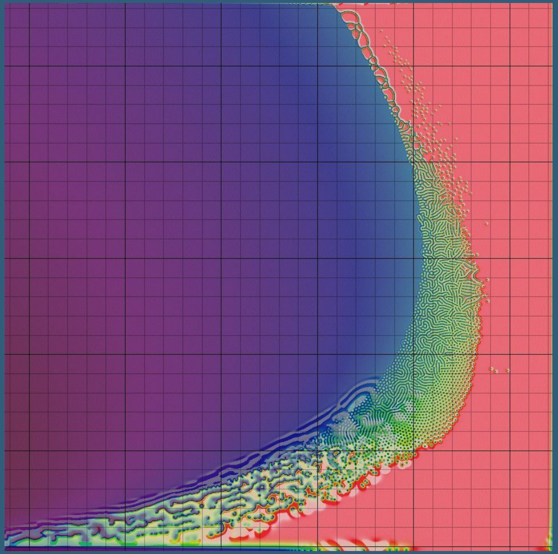


Problématique et objectifs

Afin de reconnaitre quel espèce de poisson est dans une image on étudiera la morphogenèse et les moments géométriques puis on concevra un algorithme Python permettant de reconnaitre ceux-ci.

Équation de Gray-Scott et discrétisation

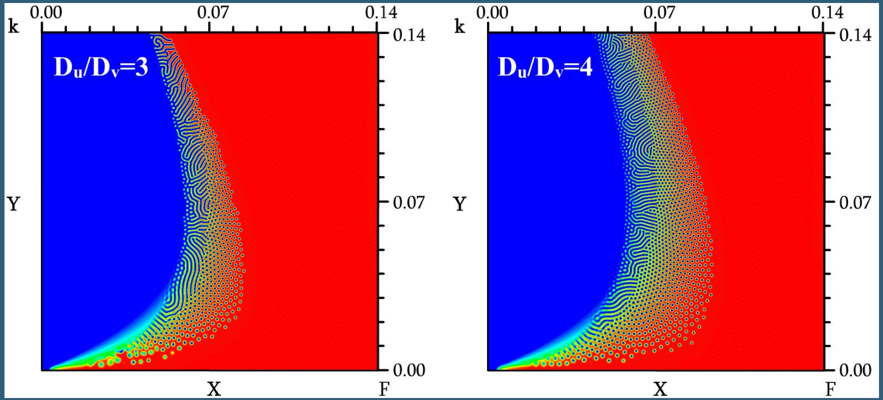
$$\frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + F(1-u),$$
$$\frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (F+k)v.$$



f

k

Discrétisation Informatique



2D filter: $\mathbf{D}^2_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

$$\Delta x = \Delta y = h$$

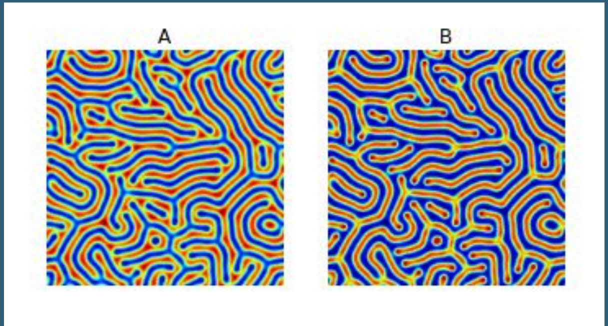
$$\Delta f(x,y) \approx \frac{f(x-h,y)+f(x+h,y)+f(x,y-h)+f(x,y+h)-4f(x,y)}{h^2}$$

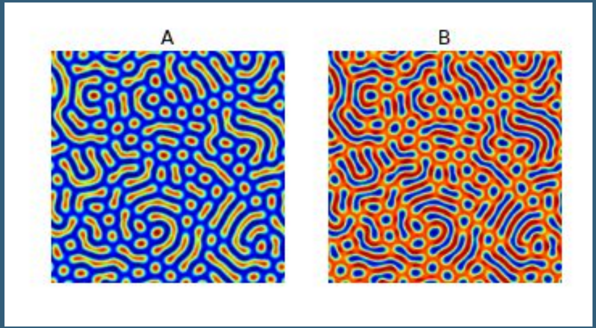
Modélisation Informatique

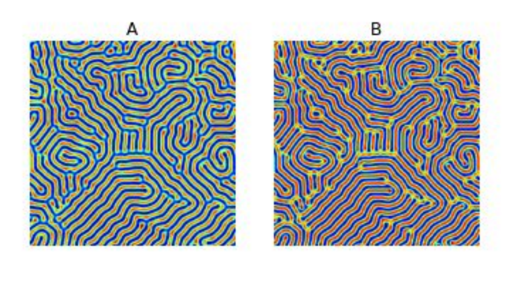
```python
import numpy as np
import matplotlib.pyplot as pl

#exemple de matrice A, tel que [[0, 1,0]
#                               [[1,-4,1]
#                                [0, 1,0]] Laplacian discret
A = np.ones((3,3))

def discrete_laplacian(M):
    L = -4*M
    L += np.roll(M, (0,-1), (0,1)) # right neighbor
    L += np.roll(M, (0,+1), (0,1)) # left neighbor
    L += np.roll(M, (-1,0), (0,1)) # top neighbor
    L += np.roll(M, (+1,0), (0,1)) # bottom neighbor
    return L

#def discrete_laplacian(Z):
#    return sp.filters.laplace(Z)

def gray_scott_update(A, B, DA, DB, f, k, delta_t):
    #Mise a jour des concentration, application de la formule Gray-Scott avec coeff diffusion DA, et DB.
    #Le feed rate et le kill rate.
    # LA et LB laplacien discret
    LA = discrete_laplacian(A)
    LB = discrete_laplacian(B)

    # Gray-Scott formula
    diff_A = (DA*LA - A*B**2 + f*(1-A)) * delta_t
    diff_B = (DB*LB + A*B**2 - (k+f)*B) * delta_t

    A += diff_A
    B += diff_B

    return A, B

def get_initial_configuration(N, random_influence=0.2):
    #On crée les matrices avec 1 et 0, puis on ajoute du bruit pour ameliorer la diffusion
    #puis on crée le gros carré au centre

    # Chaque pixel est egal a une concentration
    # a contient le u, puis le B contient le v de maniere que
    # u + 2v -> 3v
    A = (1-random_influence) * np.ones((N,N)) + random_influence * np.random.random((N,N))
    B = random_influence * np.random.random((N,N))

    N2 = N//2
    r = int(N/10.0)
    A[N2-r:N2+r, N2-r:N2+r] = 0.50
    B[N2-r:N2+r, N2-r:N2+r] = 0.25

    return A, B
```

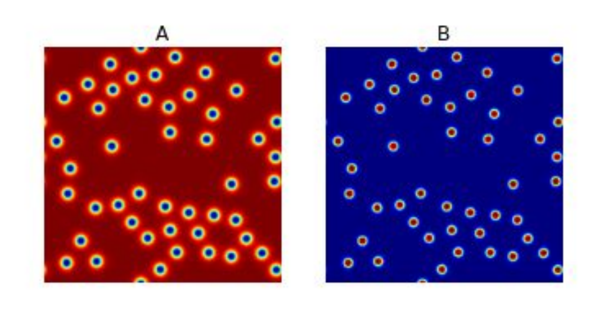
```python
def draw(A,B):
    # La partie plot du truc
    fig, ax = pl.subplots(1,2,figsize=(5.65,4))
    ax[0].imshow(A, cmap='jet')
    ax[1].imshow(B, cmap='jet')
    ax[0].set_title('A')
    ax[1].set_title('B')
    ax[0].axis('off')
    ax[1].axis('off')


# update in time
delta_t = 1.0

# Diffusion coefficients
DA = 0.19
DB = 0.05

## taux aleatoires de feed/kill
#f = rd.uniform(0.1,0.01)
#k = rd.uniform(0.045,0.07)

f=0.06
k=0.062

# taille
N = 200

# nombre de fois
N_simulation_steps = 10000


A, B = get_initial_configuration(200)

for t in range(N_simulation_steps):
    A, B = gray_scott_update(A, B, DA, DB, f, k, delta_t)
#    if t%500==0:

draw(A,B)
```

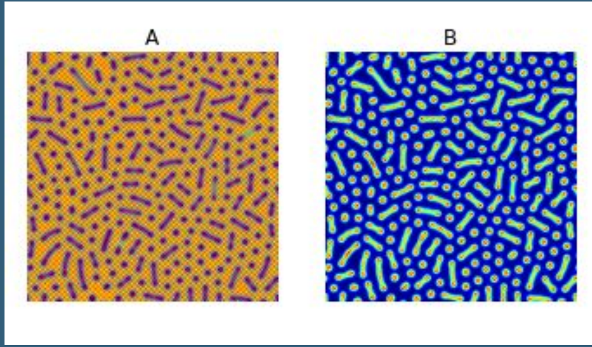
Résultats



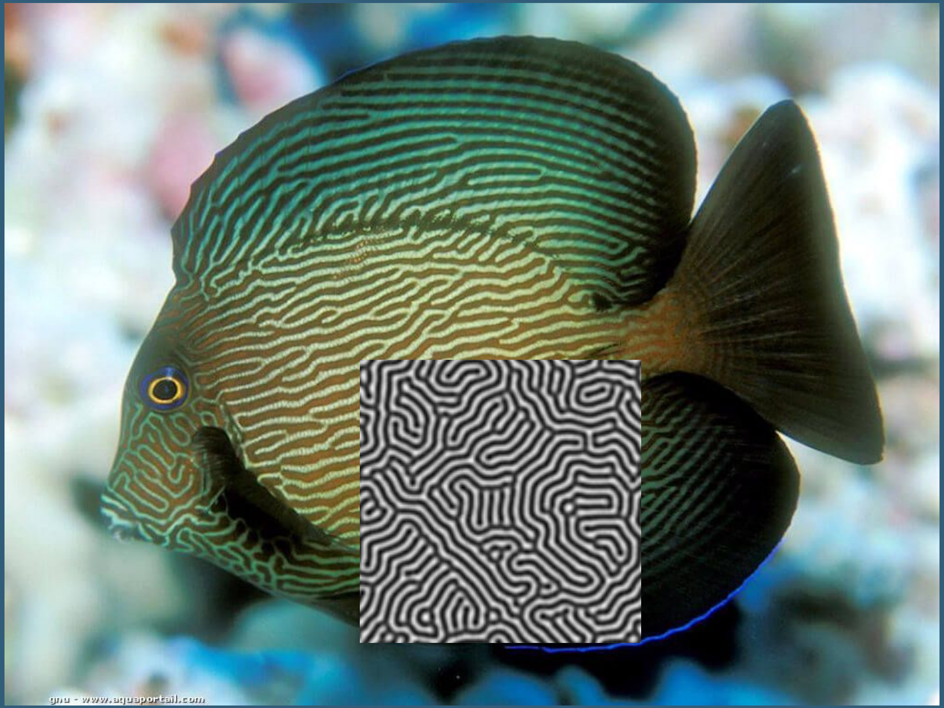


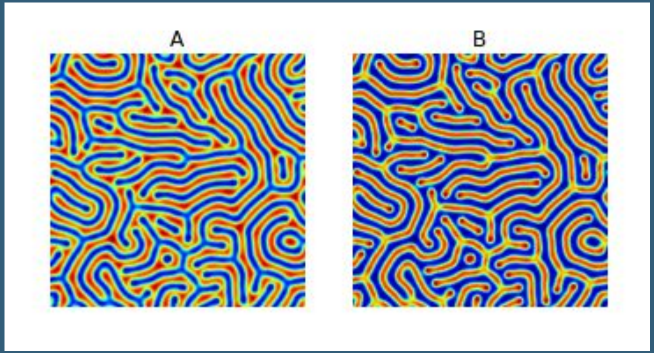


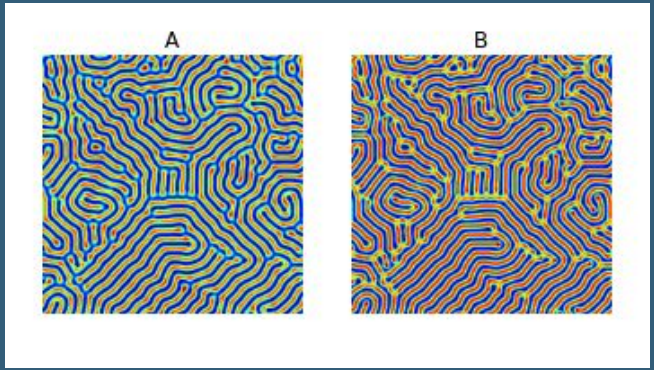




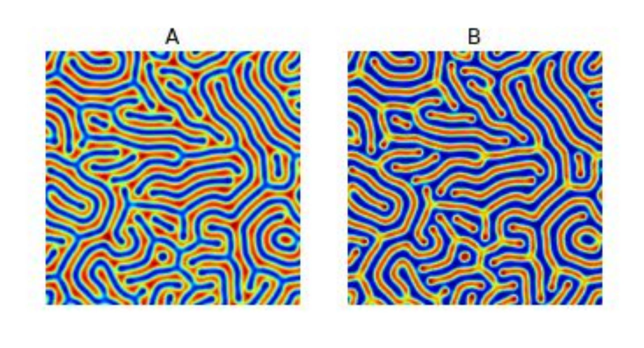
Modélisation Informatique et réalité

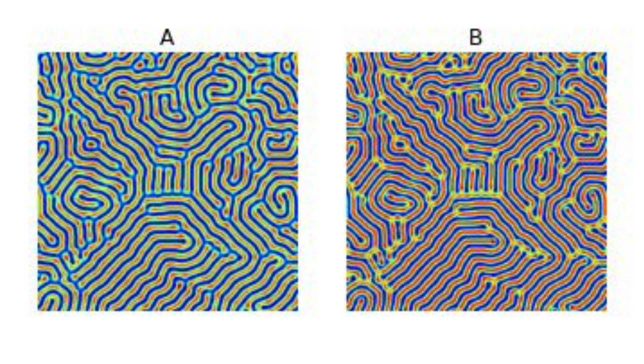






Modélisation Informatique et réalité





Moments de Hu

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

$$\bar{x} = \frac{M_{10}}{M_{00}} \text{ and } \bar{y} = \frac{M_{01}}{M_{00}}$$

$$\begin{aligned}
\mu_{00} &= M_{00}, \\
\mu_{01} &= 0, \\
\mu_{10} &= 0, \\
\mu_{11} &= M_{11} - \bar{x} M_{01} = M_{11} - \bar{y} M_{10}, \\
\mu_{20} &= M_{20} - \bar{x} M_{10}, \\
\mu_{02} &= M_{02} - \bar{y} M_{01}, \\
\mu_{21} &= M_{21} - 2\bar{x} M_{11} - \bar{y} M_{20} + 2\bar{x}^2 M_{01}, \\
\mu_{12} &= M_{12} - 2\bar{y} M_{11} - \bar{x} M_{02} + 2\bar{y}^2 M_{10}, \\
\mu_{30} &= M_{30} - 3\bar{x} M_{20} + 2\bar{x}^2 M_{10}, \\
\mu_{03} &= M_{03} - 3\bar{y} M_{02} + 2\bar{y}^2 M_{01}.
\end{aligned}$$

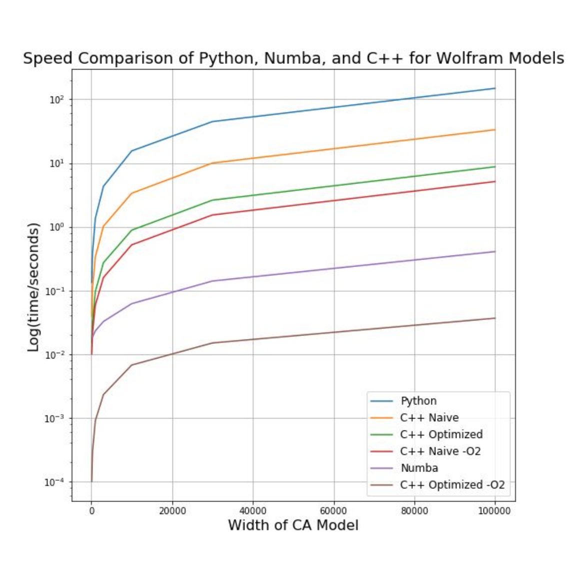
Moments de Hu

```python
def M(i,j):
    s=0
    x=len(t[0])
    y=len(t)
    for k in range (x-1):
        for l in range (y-1):
            s+=(k**i)*(l**j)*t[l][k]
    return s

def n(i,j):
    return ((u(i,j))/((u(0,0))**(1+(i+j)/2)))

def x0():
    return (m[1][0])/(m[0][0])

def y0() :
    return (m[0][1])/(m[0][0])

def u (i,j):
    s=0
    x=len(t[0])
    y=len(t)
    for k in range (x-1):
        for l in range (y-1):
            s+=((k-x0())**i)*((l-y0())**j)*t[l][k]
    return s
```

```python
def tablM():
    m=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
    m[0][0]=M(0,0)
    m[0][1]=M(0,1)
    m[1][1]=M(1,1)
    m[0][2]=M(0,2)
    m[2][1]=M(2,1)
    m[1][2]=M(1,2)
    m[0][3]=M(0,3)
    m[1][0]=M(1,0)
    m[2][0]=M(2,0)
    m[3][0]=M(3,0)
    return m


#fonction calculant les moment de Hu
def Hu ():
   I1=  (n(2,0)) + (n(0,2))
   I2= ((n(2,0))-(n(0,2)))**2 + 4*((n(1,1))**2)
   I3= (n(3,0)-3*(n(1,2)))**2 +(3*(n(2,1))-(n(0,3)))
   I4= (n(3,0)+(n(1,2)))**2+(n(2,1)+ n(0,3))**2
   I5=(n(3,0)-3*n(1,2))*(n(3,0)+n(1,2))*((n(3,0)+n(1,2))**2-3*(n(2,1)+n(0,3))**2)+(
   I6= (n(2,0)-n(0,2))*((n(3,0)+n(1,2))**2-(n(2,1)+n(0,3))**2)+4*n(1,1)/(n(3,0)+n(1
   I7= (3*n(2,1)-n(0,3))*(n(3,0)+n(1,2))*((n(3,0)+n(1,2))**2-3*(n(2,1)+n(0,3))**2)-
   I8= n(1,1)*((n(3,0)+n(1,2))**2 + (n(0,3)+n(2,1))**2)-(n(2,0)+n(0,2))*(n(3,0)+n(1
   hu=[str(I1),str(I2),str(I3),str(I4),str(I5),str(I6),str(I7),str(I8)]
   return hu
```

OpenCV et Numba/PyCuda

```python
def extractpictures (pathimages,pathrec):
    listepictures=getlist(pathimages)
    j=0
    for e in listepictures :
        print(e)
        image = cv2.imread(e)
        original = image.copy()
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray, (3, 3), 0)
        canny = cv2.Canny(blurred, 120, 255, 1)
        kernel = np.ones((5,5),np.uint8)
        dilate = cv2.dilate(canny, kernel, iterations=1)
    # Find contours
        cnts = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]
        # Iterate thorugh contours and filter for ROI
        image_number = 0
        for c in cnts:
            x,y,w,h = cv2.boundingRect(c)
            if h>20 and w>20 :
                cv2.rectangle(image, (x, y), (x + w, y + h), (36,255,12), 2)
                ROI = canny[y:y+h, x:x+w]
            #changer le canny si dessus contre original pour enregistrer l'image en couleur
                cv2.imwrite(pathrec+'ROI_{}.png'.format(j*1000+image_number), ROI)
                image_number += 1
        j+=1
```



Algorithme génétique

Bibliographie

- Turing, A.M: Philosophical Transactions of the Royal Society of London B. 237, No. 641 14/08/1952 p. 37-72 - https://blogs.scientificamerican.com/cocktail-party-physics/when-math-meets-nature-turingpatterns-and-form-constants/?redirect=1 consulté le 12 Mars 2019
https://www.pourlascience.fr/sd/mathematiques/des-equations-pour-de-bons-motifs-9026.php 12 Mars - https://phys.org/news/2015-12-mathematical-animal-stripes.html consulté le 19 Mars 2019 - https://www.lptl.jussieu.fr/user/lesne/Turing-preprint.pdf consulté le 26 Mars 2019 - https://media4.obspm.fr/public/M2R/appliquettes/Turing/AutomateTuring_website.html 23 Avril - http://www.staff.science.uu.nl/~frank011/Classes/complexity/Literature/Pearson.pdf 30 Avril 2019 - https://mrob.com/pub/comp/xmorphia/pearson-classes.html consulté le 7 Mai 2019 - https://en.wikipedia.org/wiki/Heat_equation consulté le 7 Mai 2019 - http://www.degeneratestate.org/posts/2017/May/05/turing-patterns/ consulté le 7 Mai 2019 - https://mrob.com/pub/comp/xmorphia/ogl/index.html consulté le 21 Mai
https://youtu.be/8YnWWHwu4IA et https://youtu.be/ryEQYgfd5AU consultés le 28 Mai 2019