# La morphogenèse modélisée par les structures de Turing

# Sommaire

- Présentation de la morphogenèse
- Étude d’un cas simplifié des équations de réaction diffusion
- Système d’équations proposé par Turing
- Analyse des symétries et des invariances des modèles ainsi obtenus

# Présentation de la morphogenèse

## Les motifs que présente le vivant sont extrêmement variés.

*https://www.google.fr/search?q=morphogenese&source*

*https://www.google.fr/search?biw=1391&bih=681&tbm=isch&sa=1&ei=uiohW4mCBciKU9Lms6AI&q=poisson+ange&oq=poisson+ange&gs*

# Problématique

Comment analyser les motifs obtenus avec les structures proposées par Turing ?

# Étude d’un cas simplifié des équations de réaction diffusion

**L’équation de la chaleur en 2 dimensions :**

# Étude d’un cas simplifié des équations de réaction diffusion

**L’équation de la chaleur en 2 dimensions :**

$$\frac{\partial T}{\partial t} = k \Delta T$$

# Étude d'un cas simplifié des équations de réaction diffusion

**L'équation de la chaleur en 2 dimensions :**

$$\underbrace{\frac{\partial T}{\partial t}}=k\underbrace{\Delta T}$$

$$\frac{\partial T}{\partial t}=\frac{T_{i,j}^{n+1}-T_{i,j}^{n}}{\Delta t}$$

- $$\Delta T=\frac{\partial^2 T}{dx^2}+\frac{\partial^2 T}{dy^2}$$
- $$\frac{\partial^2 T}{\partial x^2}=\frac{\frac{T_{i+1,j}^{n}-T_{i,j}^{n}}{\Delta x}-\frac{T_{i,j}^{n}-T_{i-1,j}^{n}}{\Delta x}}{\Delta x}$$

# Étude d'un cas simplifié des équations de réaction diffusion

## L'équation de la chaleur en 2 dimensions :

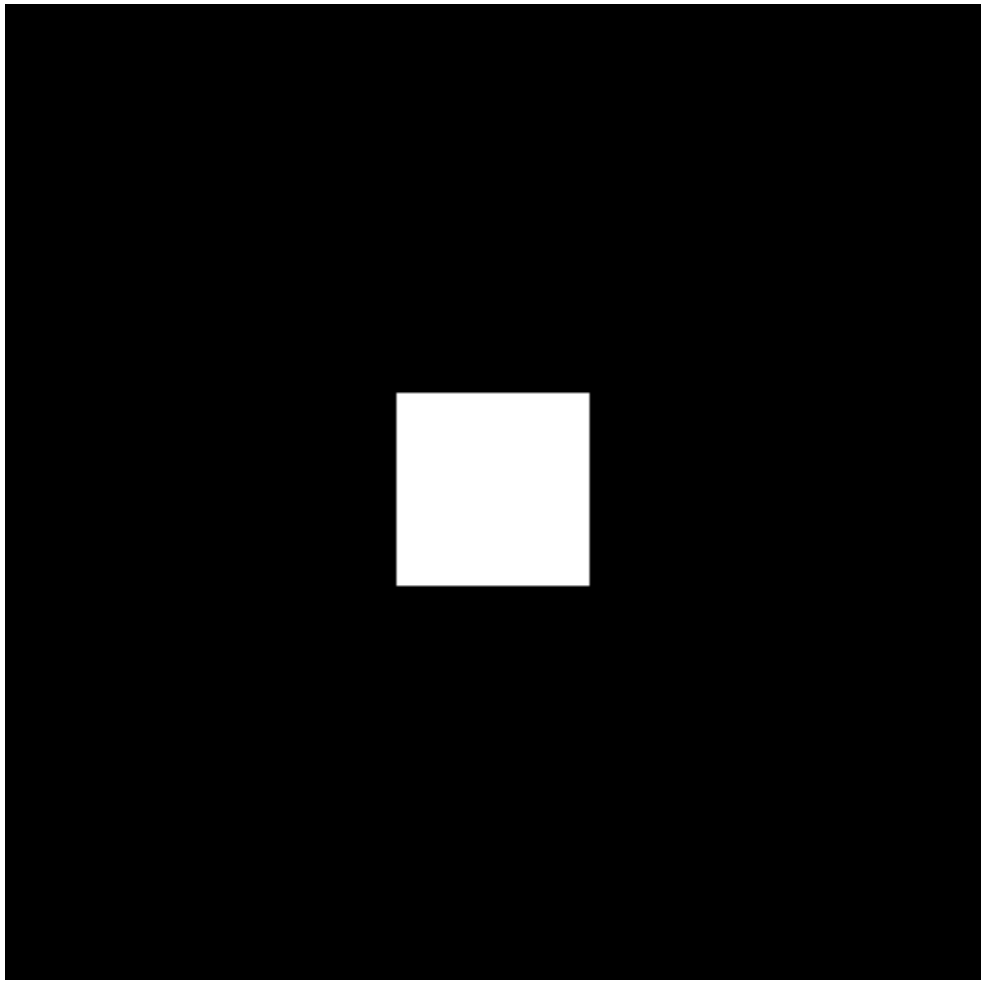
$$\frac{\partial T}{\partial t} = k \Delta T$$

$$\frac{\partial T}{\partial t} = \frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t}$$

- $$\Delta T = \frac{\partial^2 T}{dx^2} + \frac{\partial^2 T}{dy^2}$$
- $$\frac{\partial^2 T}{\partial x^2} = \frac{\frac{T_{i+1,j}^{n} - T_{i,j}^{n}}{\Delta x} - \frac{T_{i,j}^{n} - T_{i-1,j}^{n}}{\Delta x}}{\Delta x}$$

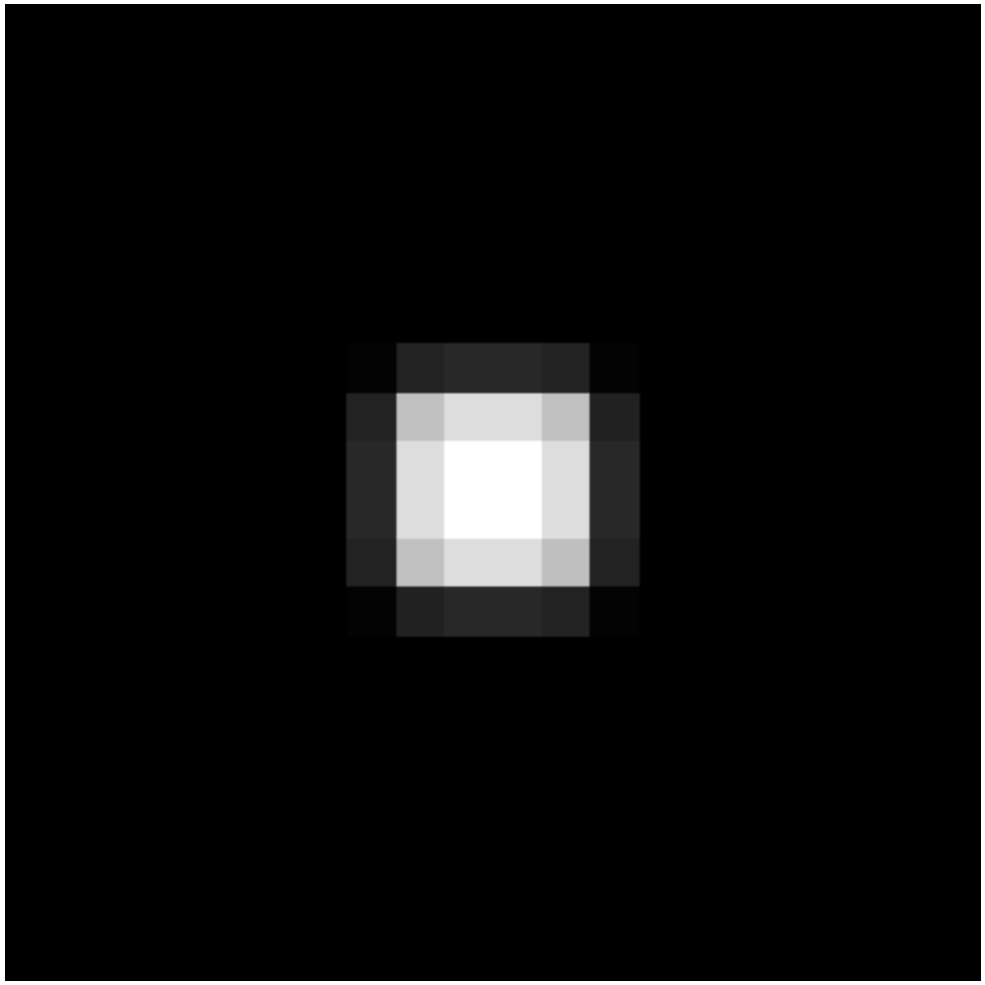
$$T_{i,j}^{n+1} = T_{i,j}^{n} + \Delta t * k * \left( \frac{1}{\Delta x^2} \left( T_{i+1,j}^{n} - 2T_{i,j}^{n} + T_{i-1,j}^{n} \right) + \frac{1}{\Delta y^2} \left( T_{i,j+1}^{n} - 2T_{i,j}^{n} + T_{i,j-1}^{n} \right) \right)$$

# Modélisation pour une plaque de métal (fer) :



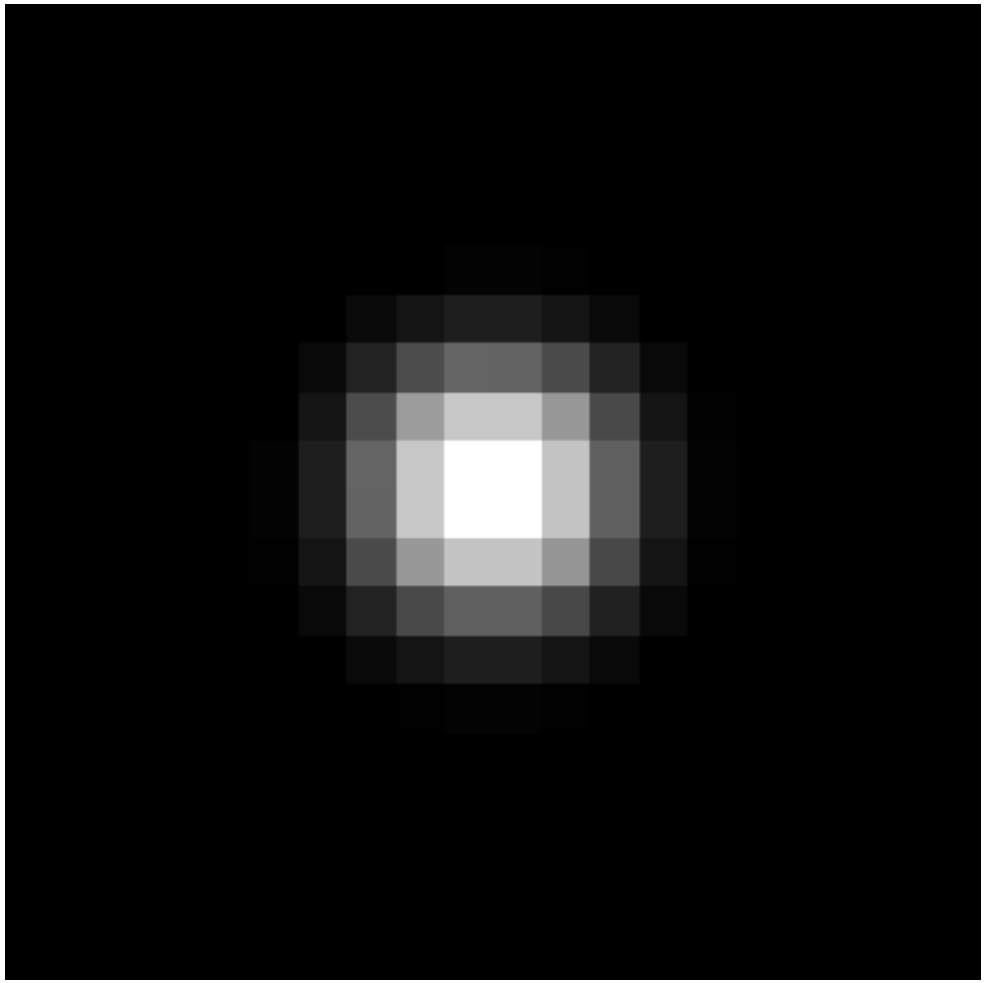
t=0s

# Modélisation pour une plaque de métal (fer) :



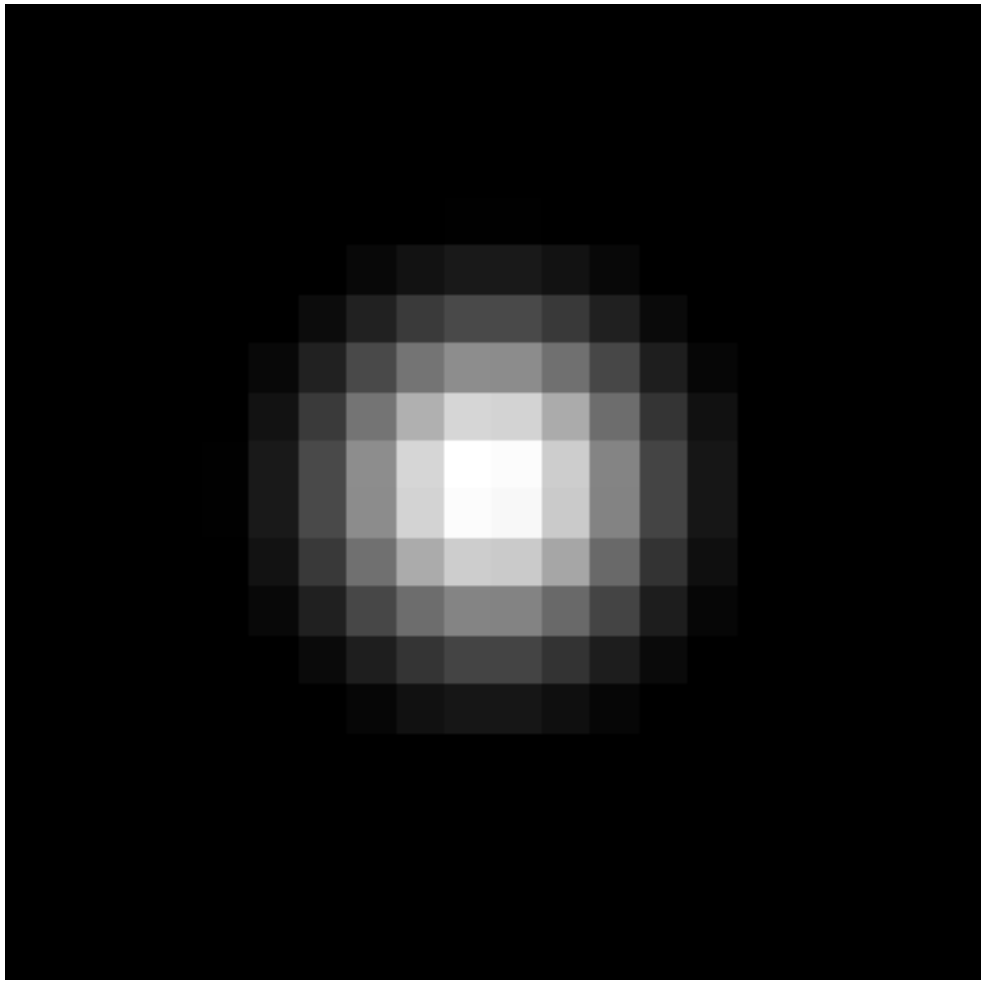
t=10s

# Modélisation pour une plaque de métal (fer) :



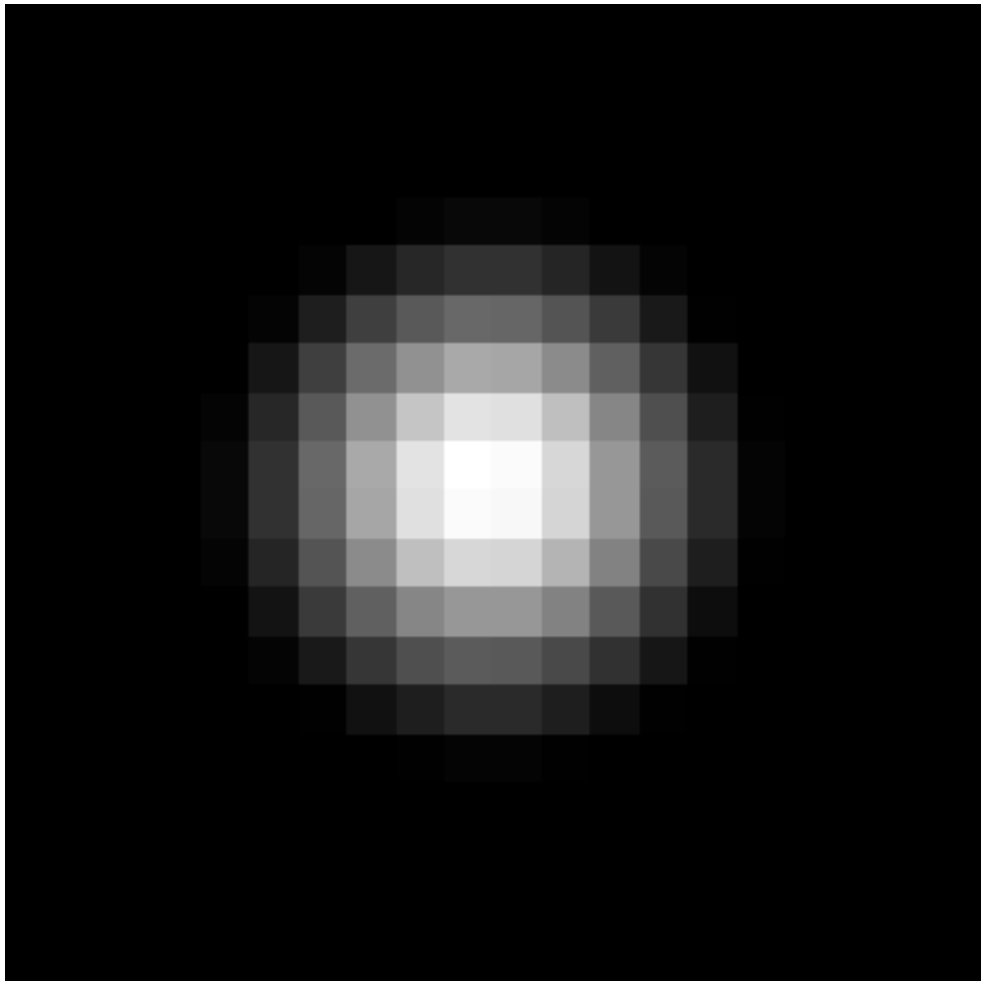
t=20s

# Modélisation pour une plaque de métal (fer) :



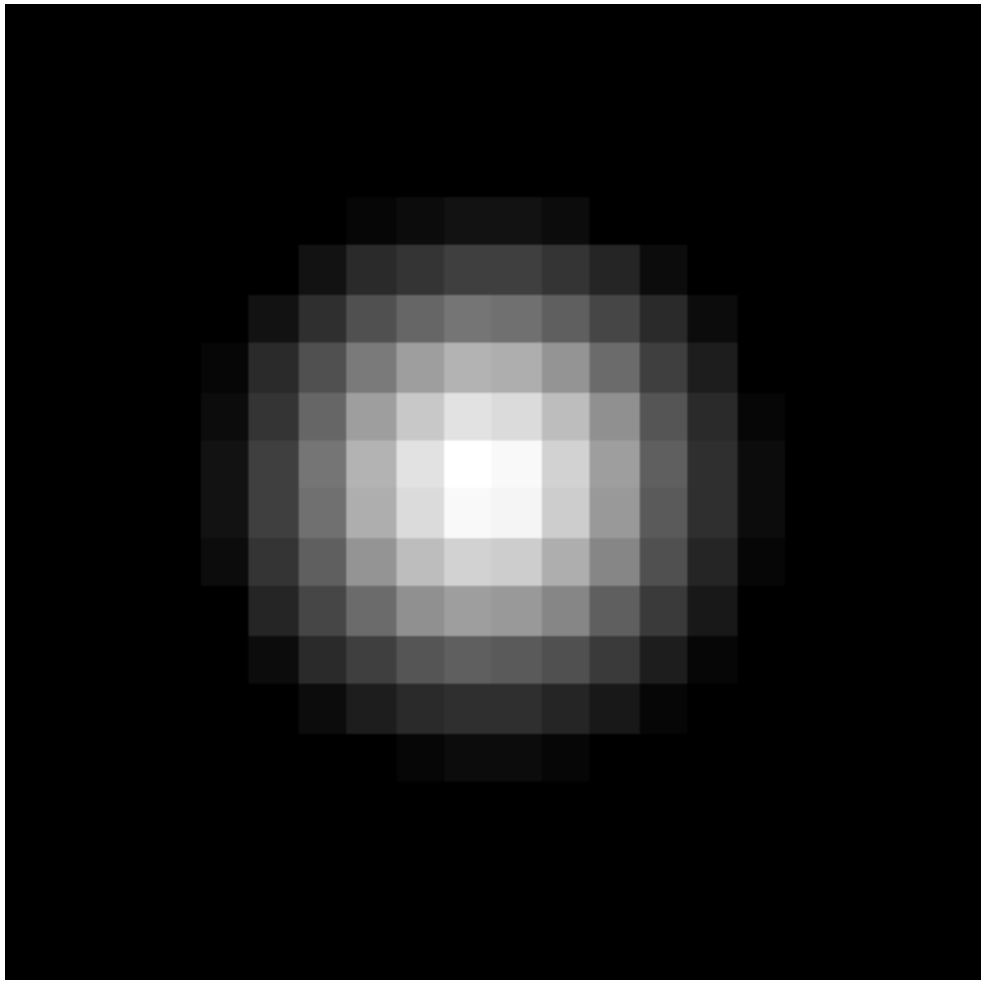
t=30s

# Modélisation pour une plaque de métal (fer) :



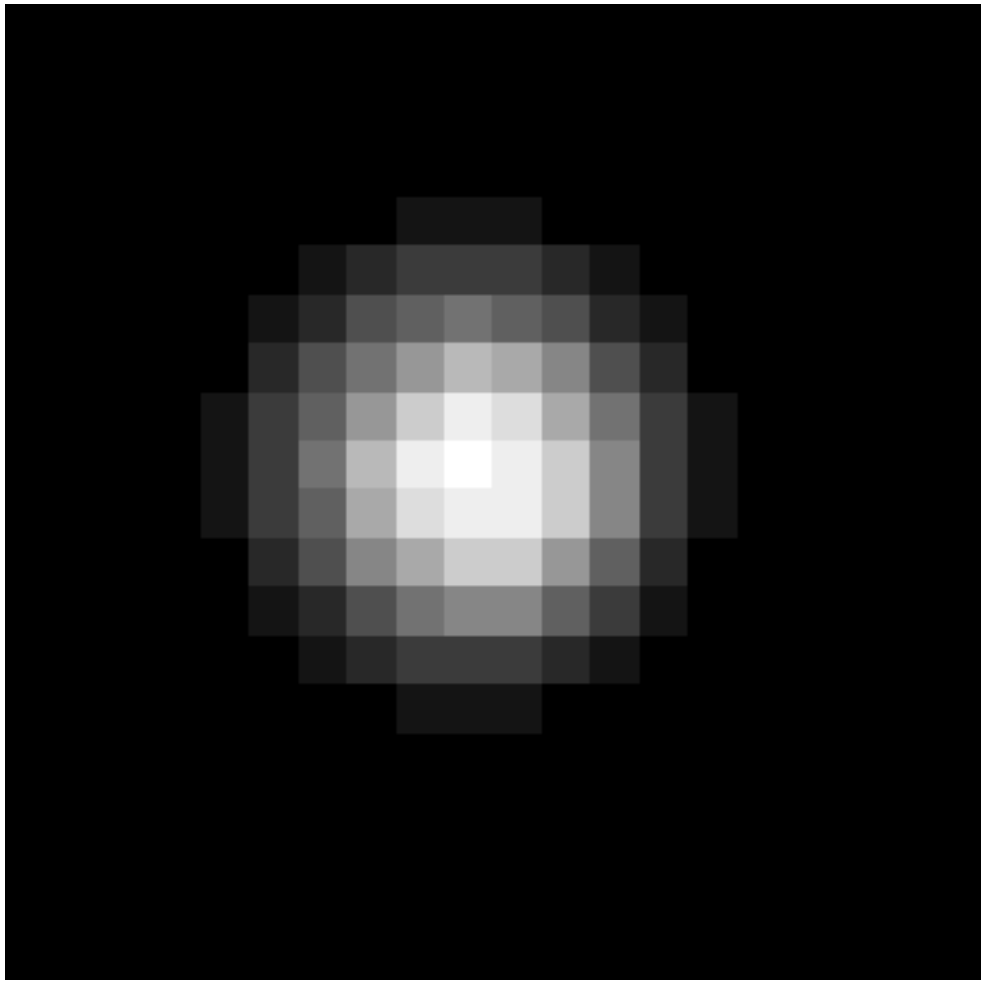
t=40s

# Modélisation pour une plaque de métal (fer) :



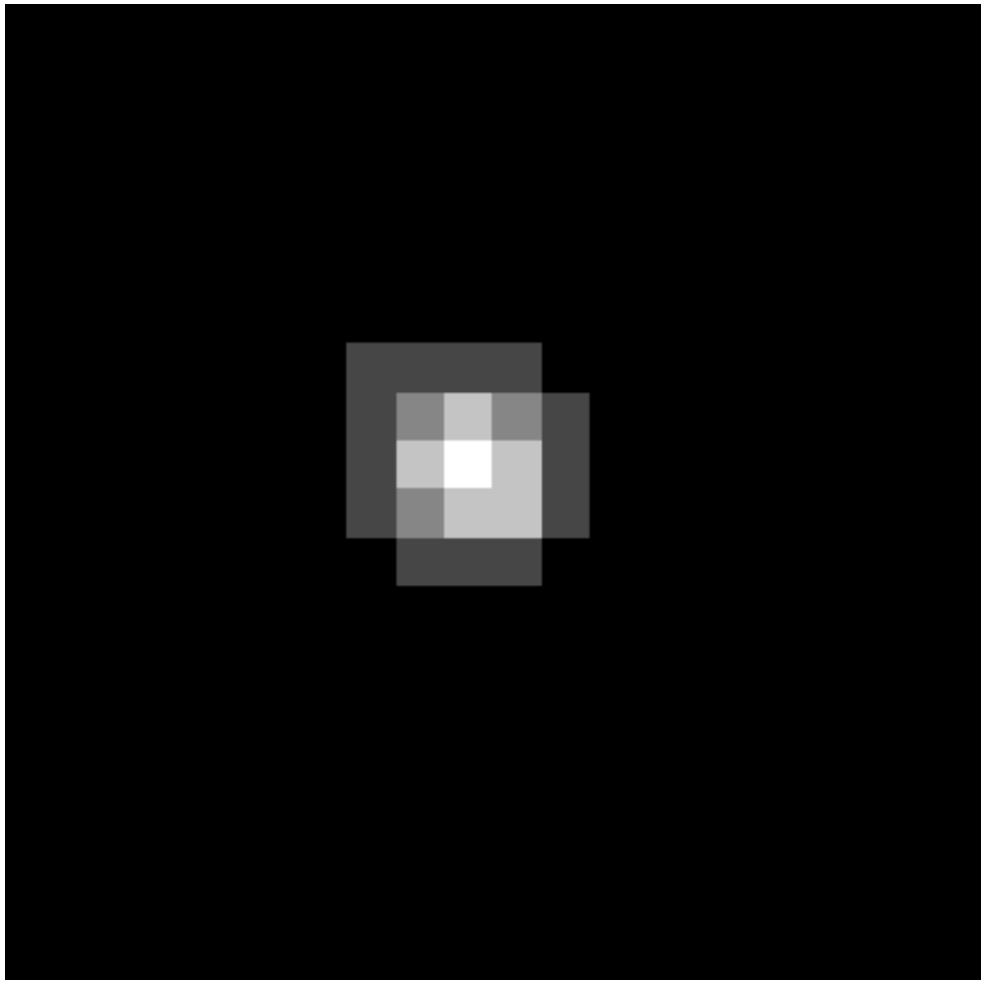
t=50s

# Modélisation pour une plaque de métal (fer) :



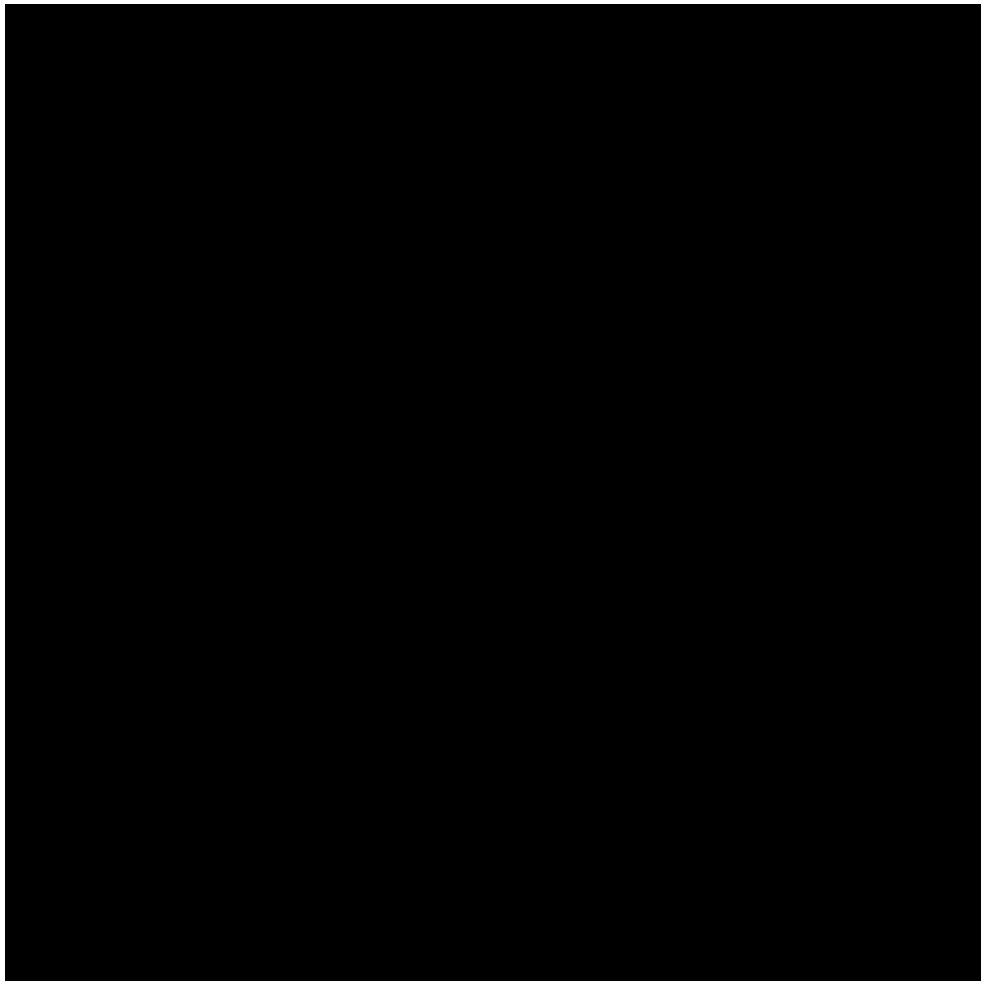
t=60s

# Modélisation pour une plaque de métal (fer) :



t=65s

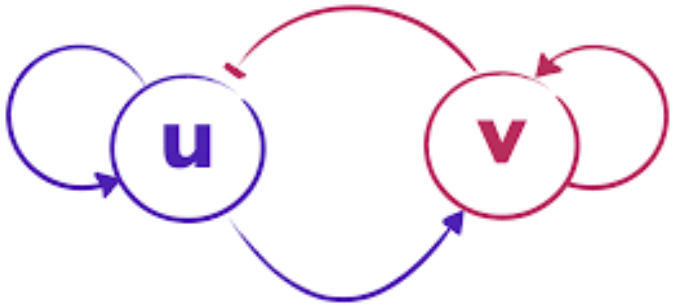
# Modélisation pour une plaque de métal (fer) :



t=70s

# Système d'équations proposé par Turing

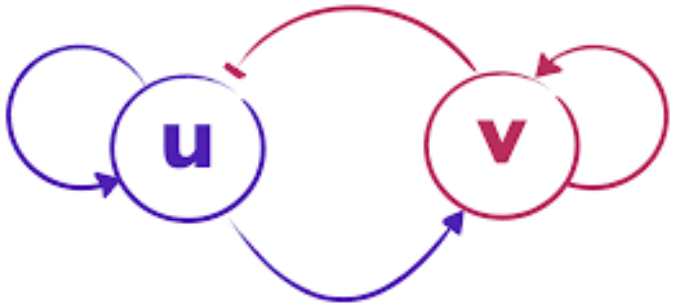
## Modélisation: Activateur-inhibiteur




Source : *http://math.univ-lyon1.fr/~pujo/TURING-IXXI.pdf*

# Système d'équations proposé par Turing

## Modélisation: Activateur-inhibiteur

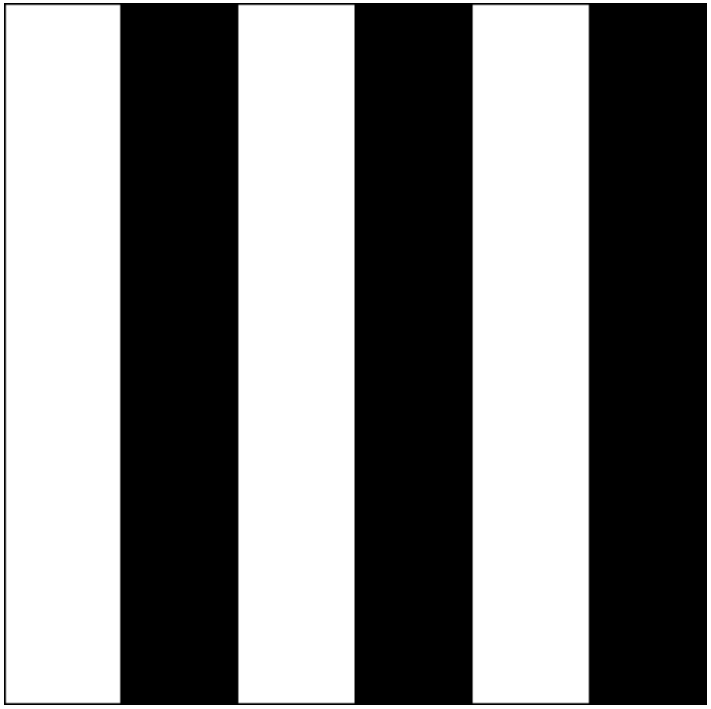


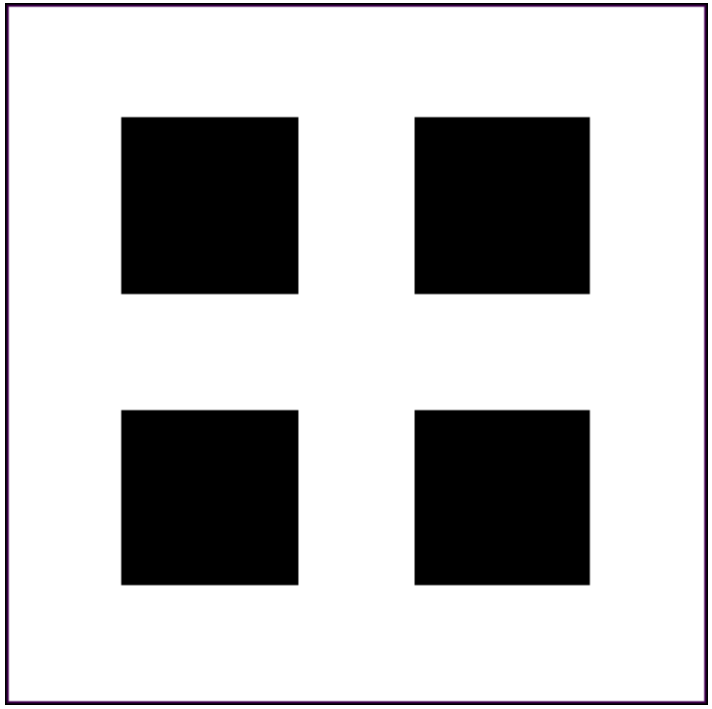

Source : *http://math.univ-lyon1.fr/~pujo/TURING-IXXI.pdf*

## Système de réaction-diffusion :

$$\begin{cases} \dfrac{\partial u}{\partial t} = D_u \Delta u + f(u,v) \\ \dfrac{\partial v}{\partial t} = D_v \Delta v + g(u,v) \end{cases}$$

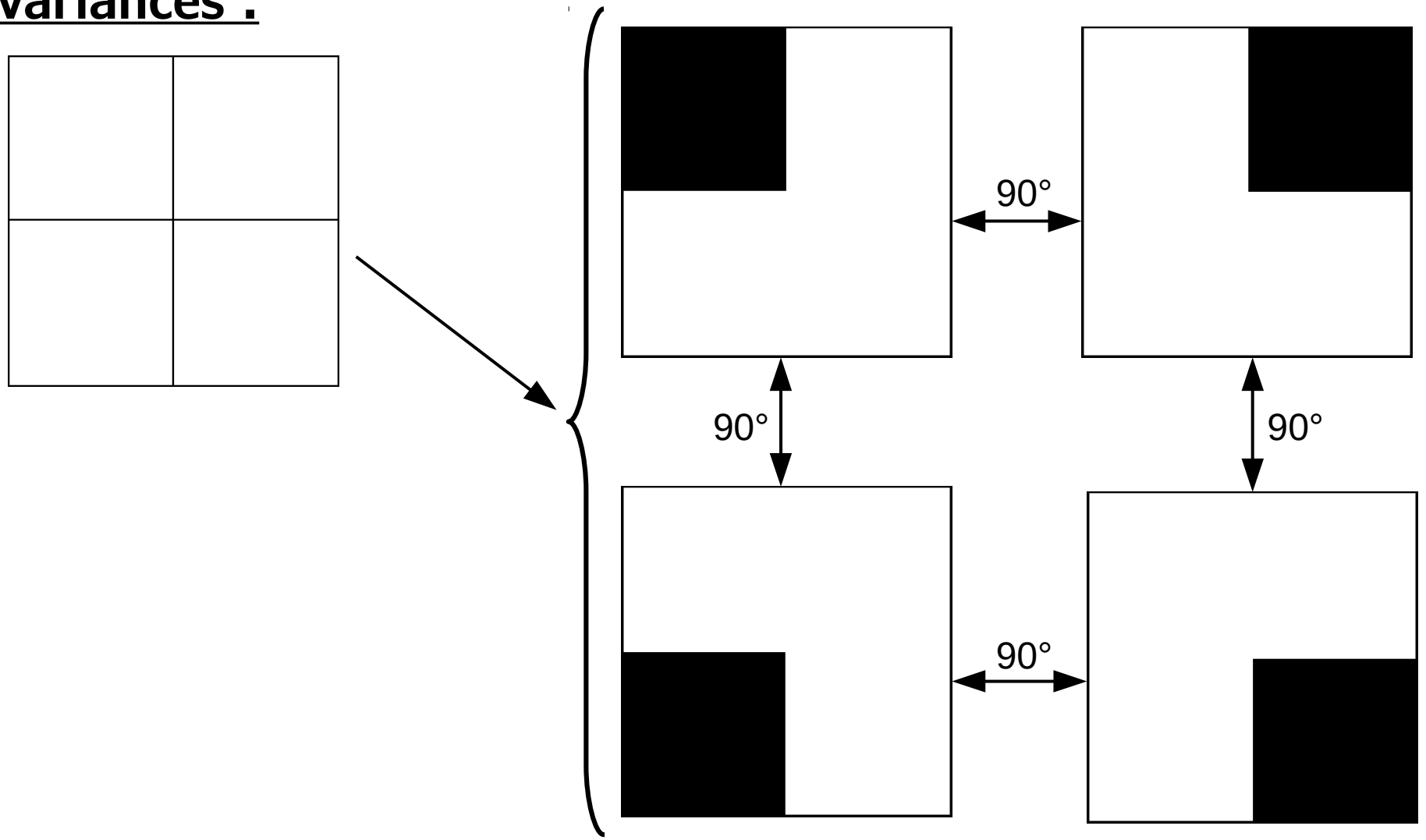
# Analyse des symétries et des invariances des modèles ainsi obtenus





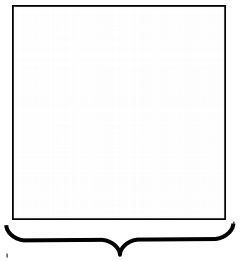
# Définitions :

## Invariances :

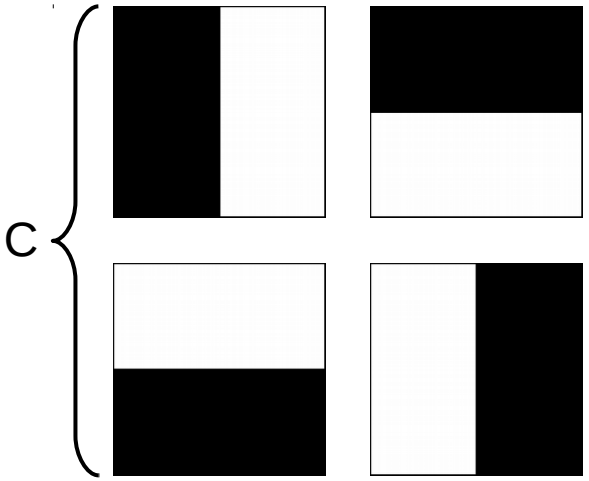


# Définitions :

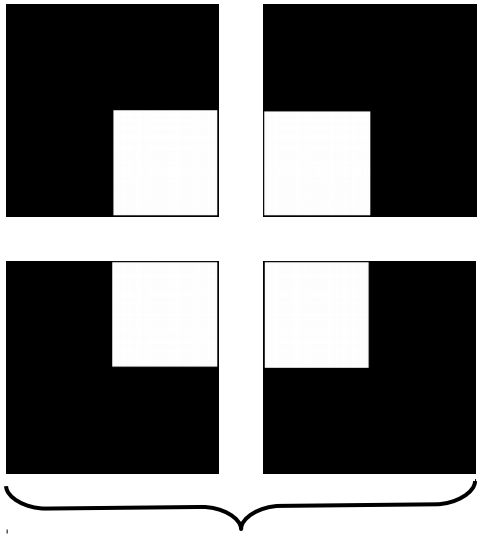
## Invariances :



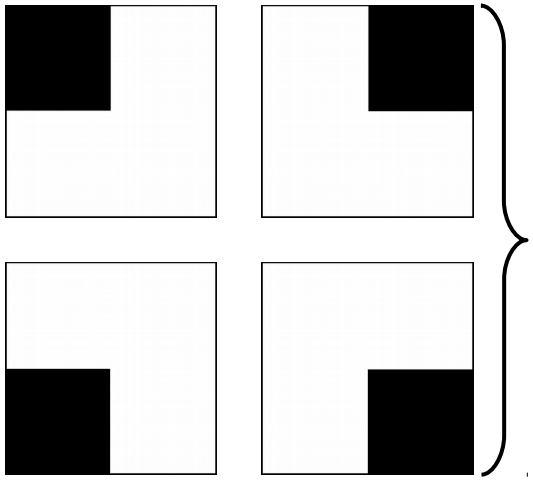
A



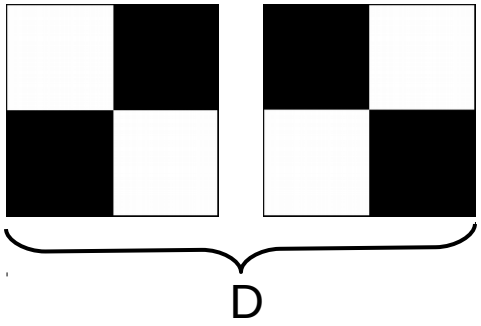
C



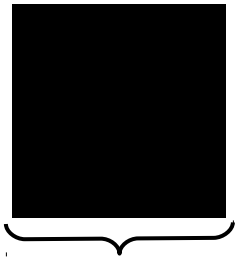
E



B



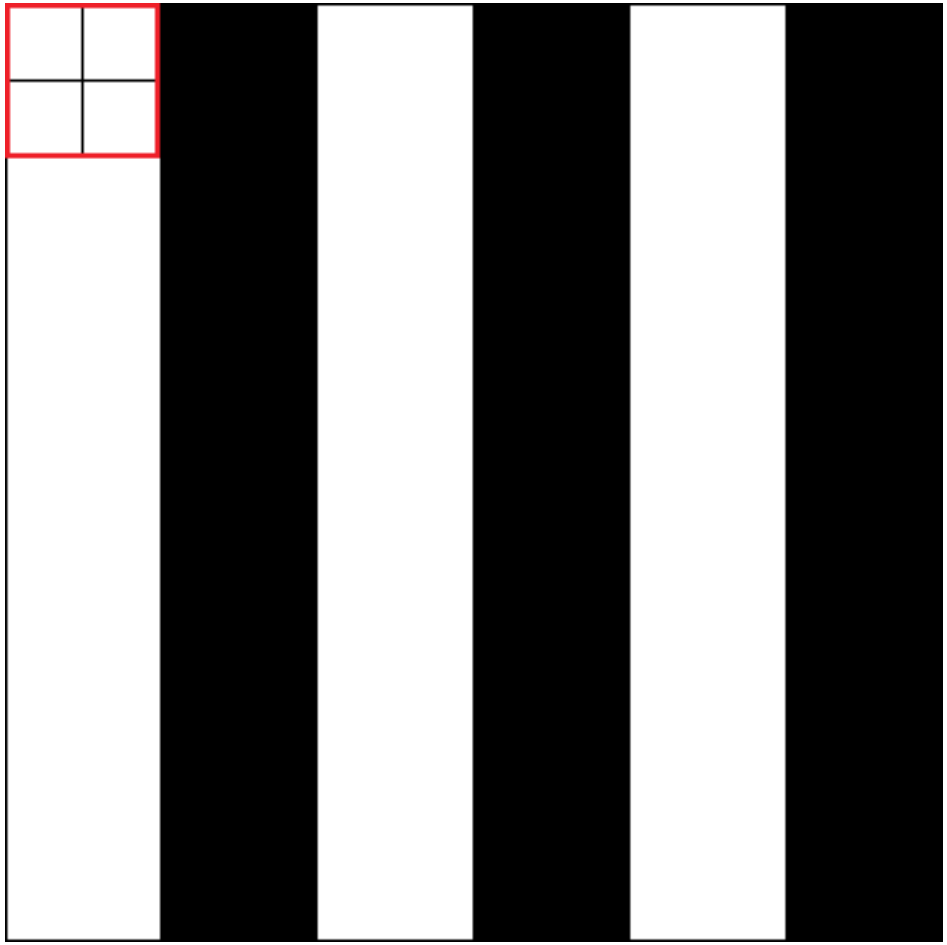
D



F

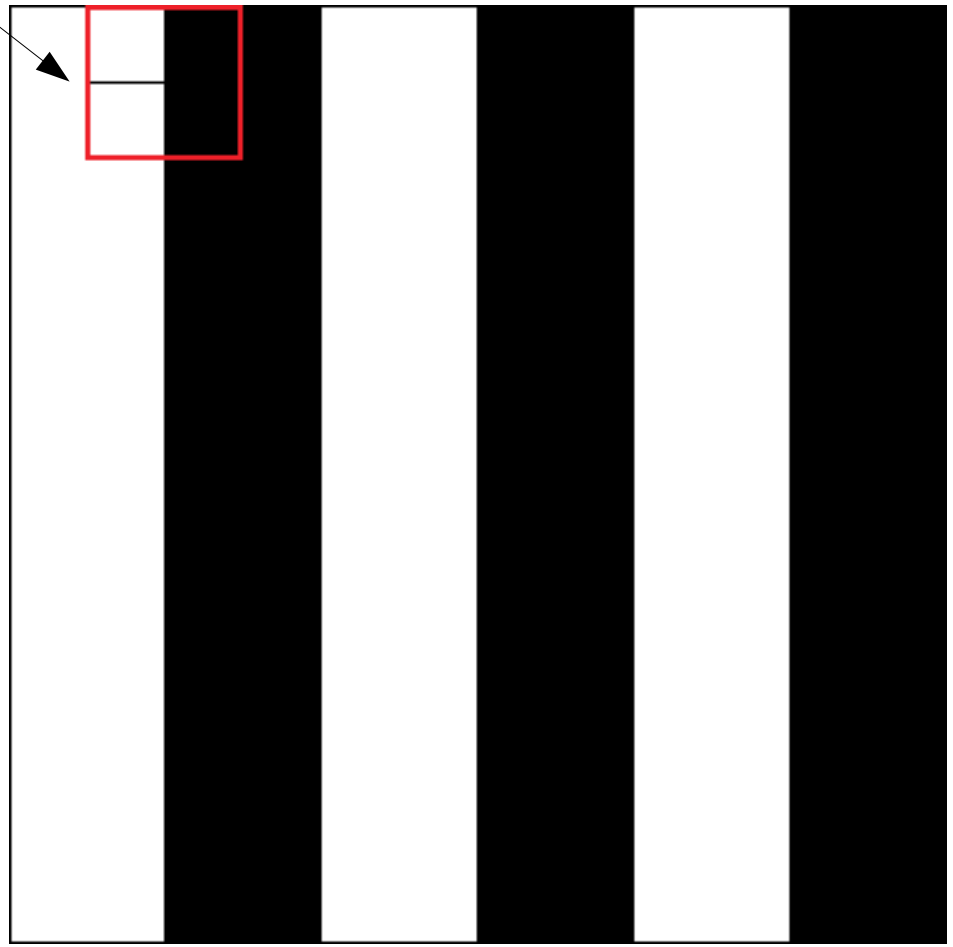
# Création d’une échelle de symétrie :

[0,0,0,0]



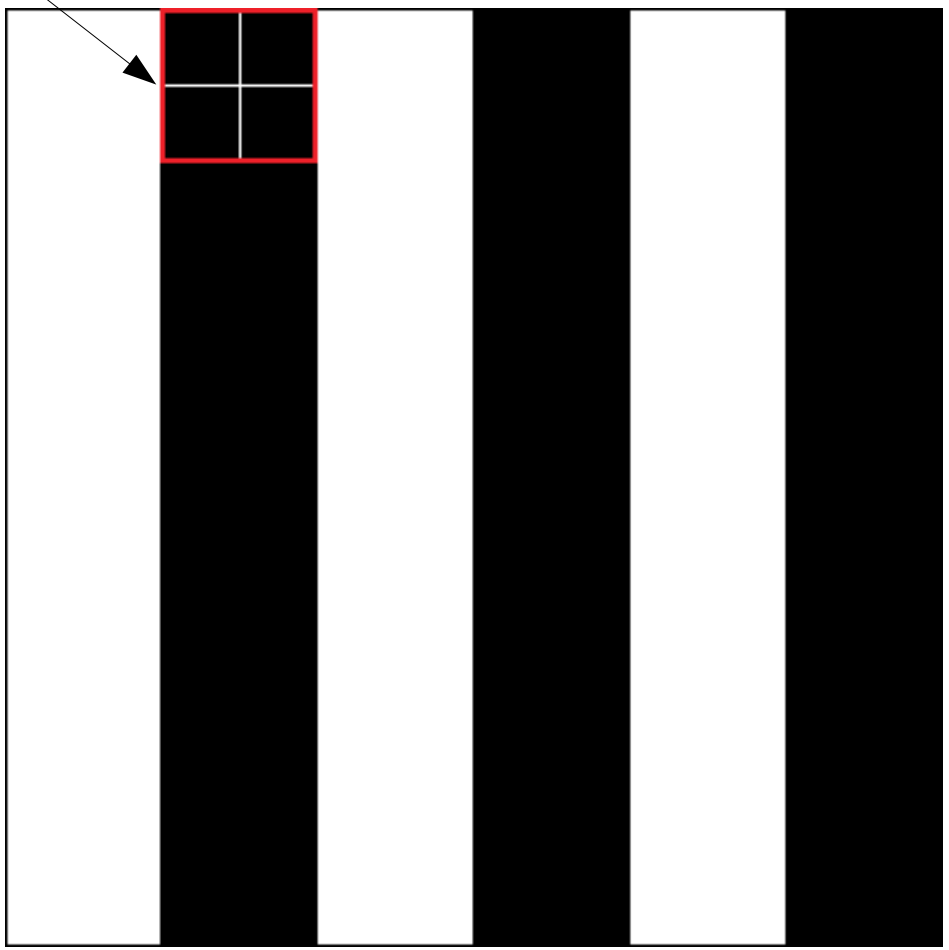
# Création d’une échelle de symétrie :

[0,1,0,1]

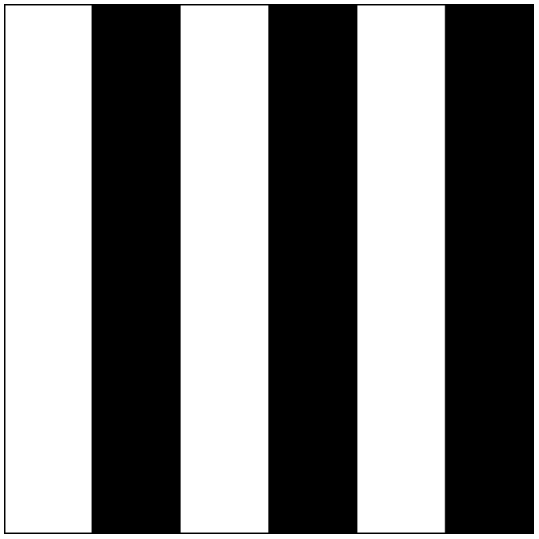


# Création d’une échelle de symétrie :

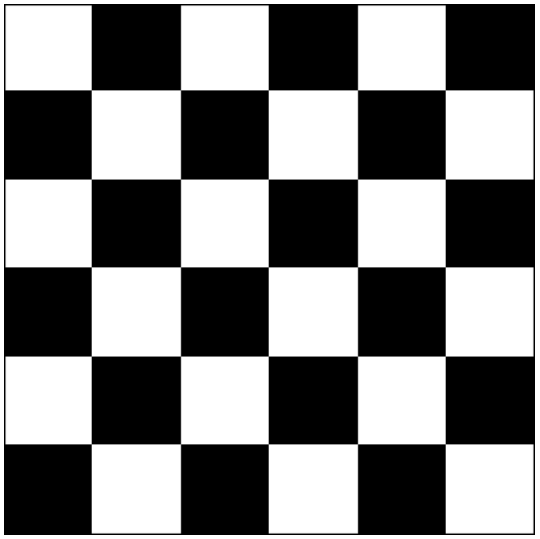
[1,1,1,1]



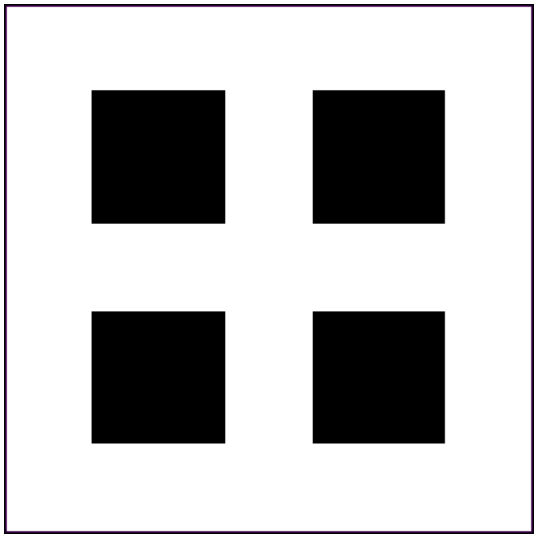
# Création d'une échelle de symétrie :



```
>>> symetrie(R)
[33, 0, 55, 0, 0, 33]
```



```
>>> symetrie(D)
[18, 0, 60, 25, 0, 18]
```



```
>>> symetrie(C)
[57, 16, 32, 0, 0, 16]
```

## Création d'une échelle de symétrie :

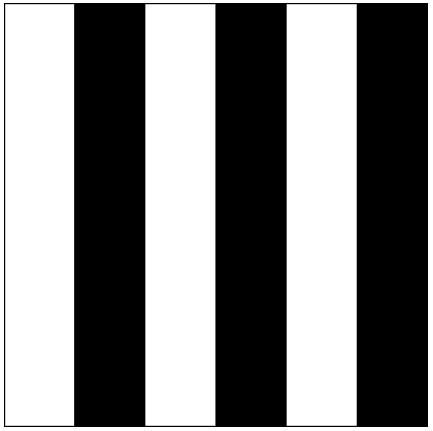
$$S=[a,b,c,d,e,f] \longrightarrow \frac{\sqrt{a^2+b^2+c^2+d^2+e^2+f^2}}{a+b+c+d+e+f}$$

# Création d'une échelle de symétrie :

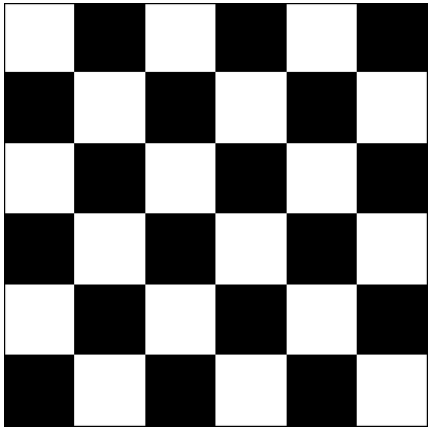
$$S=[a,b,c,d,e,f] \longrightarrow \frac{\sqrt{a^2+b^2+c^2+d^2+e^2+f^2}}{a+b+c+d+e+f}$$

## Résultats :

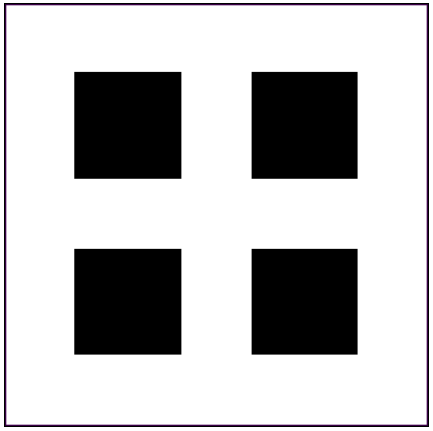
```
>>> echelle(R)
0.596130774936545 5
```



```
>>> echelle(D)
0.576916334314949 8
```

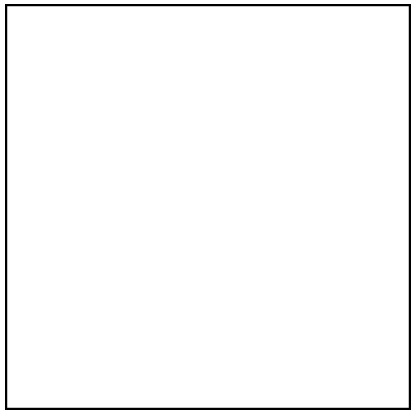


```
>>> echelle(C)
0.571683424975909 5
```



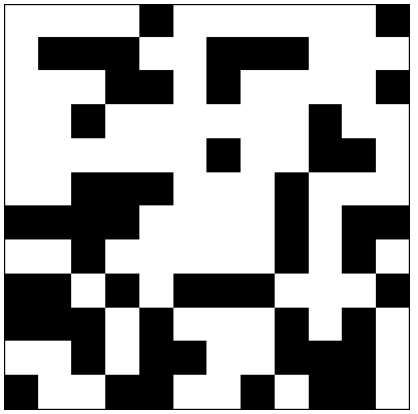
## Création d’une échelle de symétrie :

# Limites de cette première échelle :



```
>>> symetrie(Z)
[121, 0, 0, 0, 0, 0]

>>> echelle(Z)
1.0
```

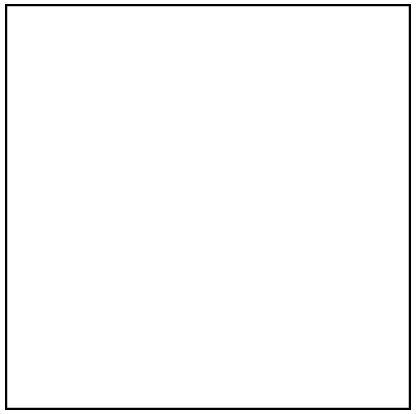


```
>>> symetrie(A)
[15, 41, 31, 14, 17, 3]

>>> echelle(A)
0.4791250591945701
```

## Création d'une échelle de symétrie :

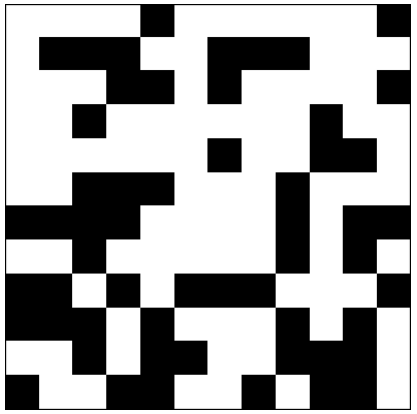
# Limites de cette première échelle :



```
>>> symetrie(Z)
[121, 0, 0, 0, 0, 0]

>>> echelle(Z)
1.0
```

$$S=[a,b,c,d,e,f]$$



```
>>> symetrie(A)
[15, 41, 31, 14, 17, 3]

>>> echelle(A)
0.4791250591945701
```

$$1+\ln\left(\frac{\sqrt{a^2+b^2+c^2+d^2+e^2+f^2}}{a+b+c+d+e+f}\right)$$

## Création d'une échelle de symétrie :

# Résultats, analyse et comparaisons :

| | A (aléatoire) | C (carrés) | D (damier) | R (rayures) | Z (zéro) |
|---|---|---|---|---|---|
| **symétrie()** | [15,41,31,14,17,3] | [57,16,32,0,0,16] | [18, 0, 60, 25, 0, 18] | [33, 0, 55, 0, 0, 33] | [121, 0, 0, 0, 0, 0] |
| **échelle()** | 0,479 | 0,572 | 0,577 | 0,596 | **1,000** |
| **echelle2()** | 0,264 | 0,441 | 0,450 | 0,483 | **1,000** |

# Création d'une échelle de symétrie :

## Résultats, analyse et comparaisons :

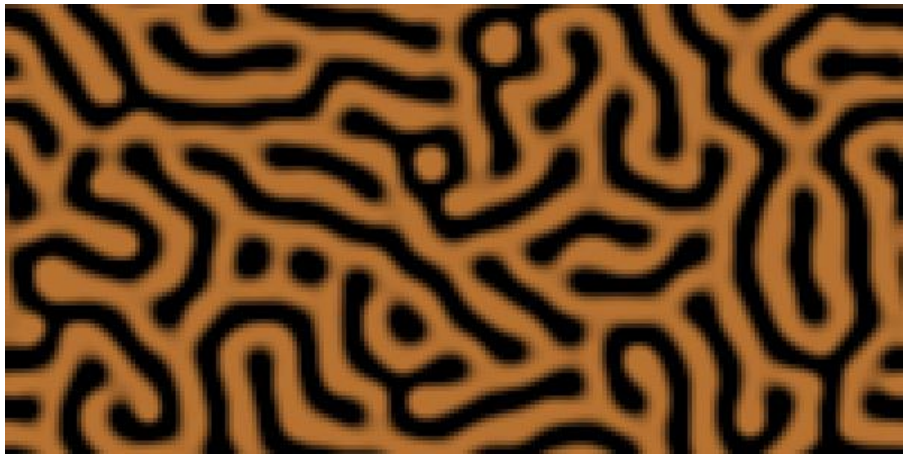
| | A (aléatoire) | C (carrés) | D (damier) | R (rayures) | Z (zéro) |
|---|---|---|---|---|---|
| **symétrie()** | [15,41,31,14,17,3] | [57,16,32,0,0,16] | [18, 0, 60, 25, 0, 18] | [33, 0, 55, 0, 0, 33] | [121, 0, 0, 0, 0, 0] |
| **échelle()** | 0,479 | 0,572 | 0,577 | 0,596 | **1,000** |
| **echelle2()** | 0,264 | 0,441 | 0,450 | 0,483 | **1,000** |

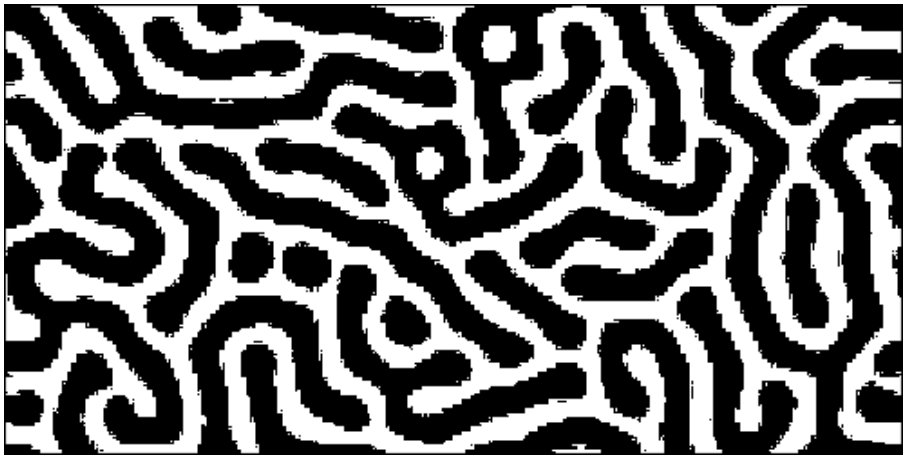
## Détermination du zéro (500*500) :

```
>>> zero(100)
0.22040786060361428
```

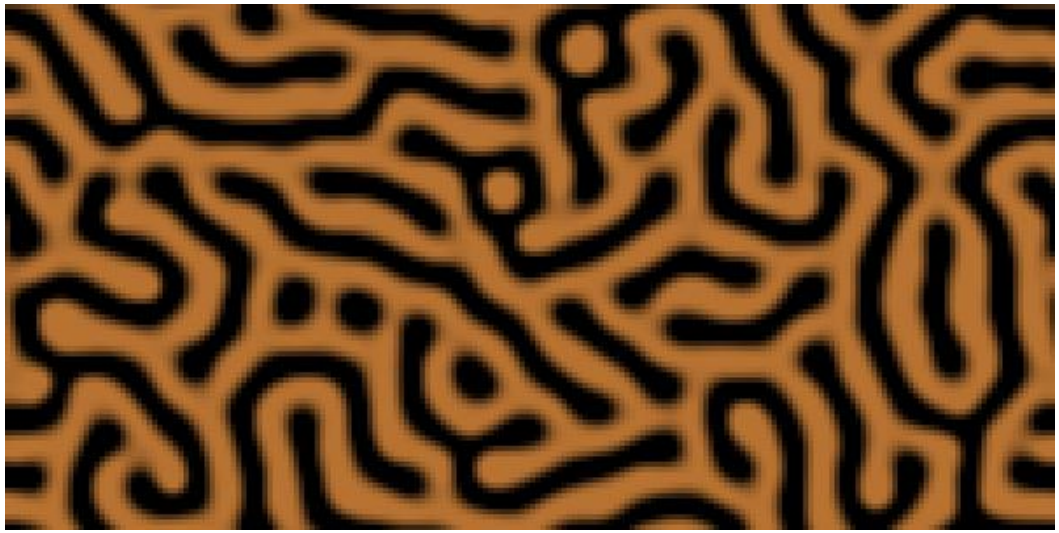
## Application à divers exemples :

Conversion des images :

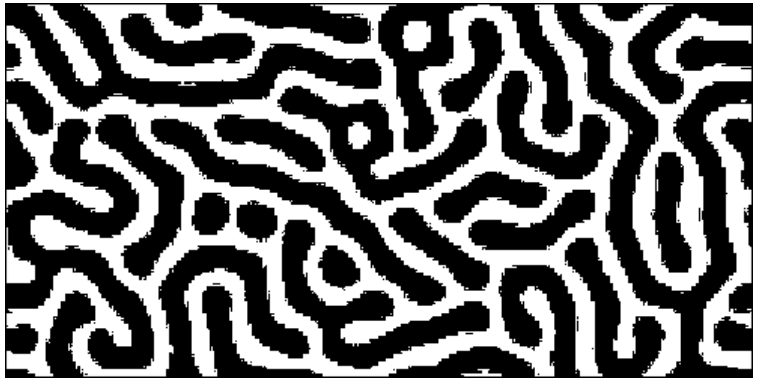




# Application à divers exemples :

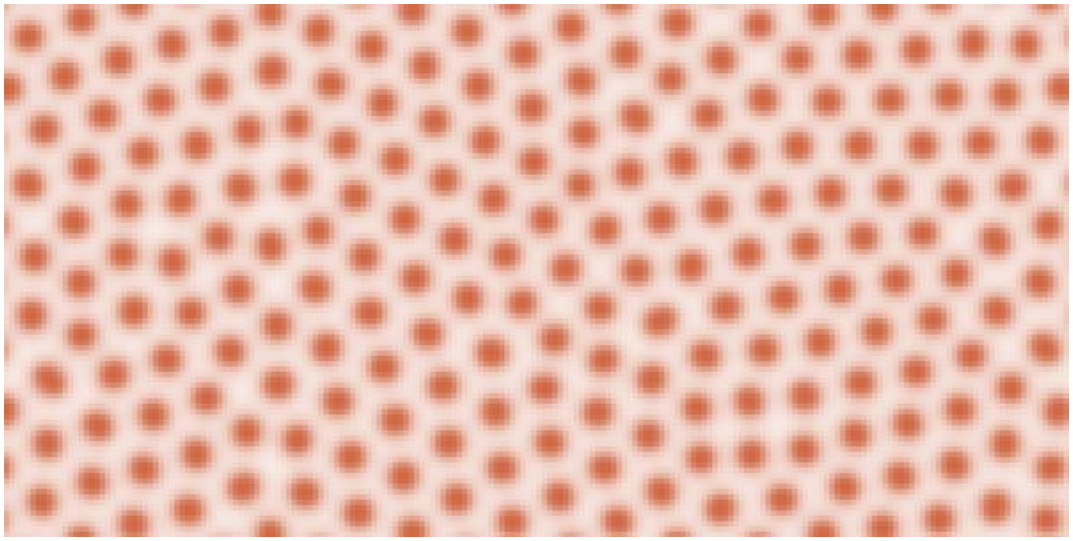


⑤

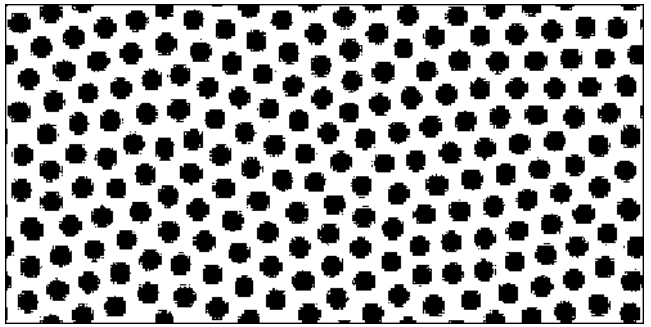


| | echelle2() |
|---|---|
| empreinte1 | 0,57029 |
| empreinte2 | 0,57031 |
| empreinte3 | 0,56965 |

# Application à divers exemples :



④



| | echelle2() |
|---|---|
| points1 | 0,586 |
| points2 | 0,581 |
| points3 | 0,584 |

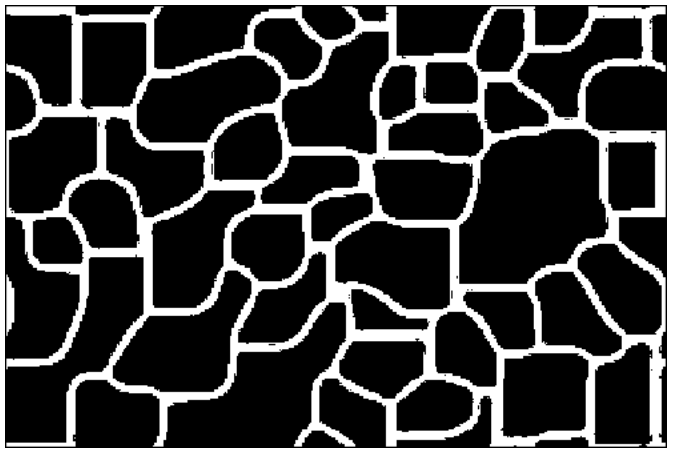
# Application à divers exemples :

③

| | echelle2() |
|---|---|
| queue1 | 0,745 |
| queue2 | 0,742 |
| queue3 | 0,744 |

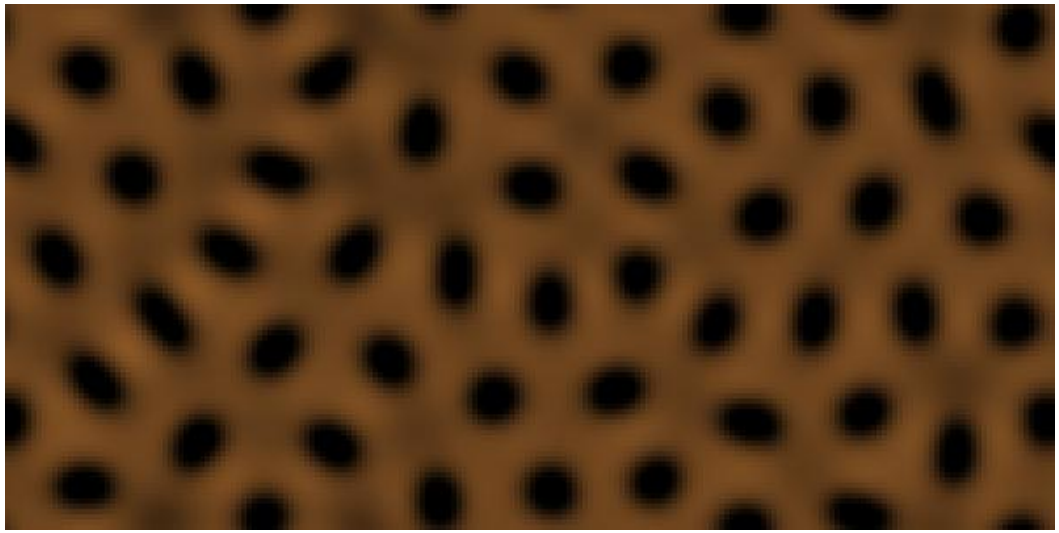
# Application à divers exemples :

②

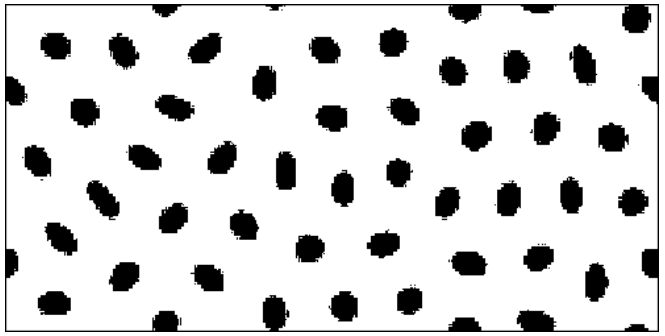


| | echelle2() |
|---|---|
| girafe1 | 0,749 |
| girafe2 | 0,766 |
| girafe3 | 0,746 |

# Application à divers exemples :



①



| | echelle2() |
|---|---|
| guépard1 | 0,828 |
| guépard2 | 0,820 |
| guépard3 | 0,822 |

# Conclusion

L’échelle de symétrie ainsi créée est valide et les écarts entre les valeurs peuvent être expliqués par :

- **dimension des images étudiées/création de l’échelle.**
- **dimension motif élémentaire très petite**

# Annexe

```
def equation_chaleur_2D(L,dt):#On suppose ici que L est une matrice carrée
    n=len(L)
    for i in range(1,n-1):
        for j in range(1,n-1):
            L[i][j] = L[i][j]+dt*k*((1/dx**2)*(L[i+1][j]-2*L[i][j]+L[i-1][j])+(1/dy**2)*(L[i][j+1]-2*L[i][j]+L[i][j-1]))
    return(L)
```

```
#definition d'une echelle de couleur(noir et blanc):
M= 200
color_list=[[i/(M-1),i/(M-1),i/(M-1)]for i in range (M)]
my_cmap=mcolors.ListedColormap(color_list,name='from_list',N=None)
```

```
def plaque(n):
    L=points_tab(n)
    for i in range(len(L)):
        for j in range(len(L)):
            if L[i][j]==0:
                L[i][j]=20
    k=n//2
    if n%2==1:
        for i in [k-2,k-1,k,k+1,k+2]:
            for j in [k-2,k-1,k,k+1,k+2]:
                L[i][j]=510
    if n%2==0:
        for i in [k-2,k-1,k,k+1]:
            for j in [k-2,k-1,k,k+1]:
                L[i][j]=510
    return(L)
```

```
def representationplaque(n,m):
    L=equation_chaleur_2D(test(n),dt)
    for i in range(m):
        L=equation_chaleur_2D(L,dt)
    plt.imshow(L)
    plt.show()
```

```
def evolution_e_c_2D(L,n):
    DT=[]
    for i in range(n):
        DT.append(dt*i)
        print(equation_chaleur_2D(L,DT[i]),'\n')
        L=equation_chaleur_2D(L,DT[i])
    plt.imshow(L,cmap=my_cmap)
    plt.show()
```

# Annexe

```
def rayurel():
    L=points_tab(12)
    n=len(L)
    for j in [2,3,6,7,10,11]:
        for i in range(n):
            L[i][j]=1
    return(L)
```

```
def carrel():
    L=points_tab(12)
    n=len(L)
    for j in [2,3,4,7,8,9]:
        for i in [2,3,4,7,8,9]:
            L[i][j]=1
    return(L)
```

```
def damier():
    L=points_tab(12)
    for j in [2,3,6,7,10,11]:
        for i in [0,1,4,5,8,9]:
            L[i][j]=1
    for j in [0,1,4,5,8,9]:
        for i in [2,3,6,7,10,11]:
            L[i][j]=1
    return(L)
```

```
def alea(n):
    L=points_tab(n)
    for i in range(n):
        for j in range(n):
            L[i][j]=rd.randint(0,1)
    return(L)
```

```
def decoupage(L):
    n=len(L)
    m=len(L[0])
    D=[]
    for i in range(n-1):
        for j in range(m-1):
            D.append([L[i][j],L[i][j+1],L[i+1][j],L[i+1][j+1]])
    return(D)
```

```
def symetrie(L):
    D=decoupage(L)
    S=[0]*6
    n=len(D)
    for i in range(n):
        if D[i]==[0,0,0,0]:
            S[0]+=1
        if D[i]==[1,0,0,0] or D[i]==[0,1,0,0] or D[i]==[0,0,0,1] or D[i]==[0,0,1,0]:
            S[1]+=1
        if D[i]==[1,1,0,0] or D[i]==[1,0,1,0] or D[i]==[0,0,1,1] or D[i]==[0,1,0,1]:
            S[2]+=1
        if D[i]==[1,0,0,1] or D[i]==[0,1,1,0]:
            S[3]+=1
        if D[i]==[1,0,1,1] or D[i]==[1,1,0,1] or D[i]==[0,1,1,1] or D[i]==[1,1,1,0]:
            S[4]+=1
        if D[i]==[1,1,1,1]:
            S[5]+=1
    return(S)
```

# Annexe

```
def echelle(L):
    S=symetrie(L)
    return(sqrt(S[0]**2+S[1]**2+S[2]**2+S[3]**2+S[4]**2+S[5]**2)/(S[0]+S[1]+S[2]+S[3]+S[4]+S[5]))

def echelle2(L):
    S=symetrie(L)
    return(1+log(sqrt(S[0]**2+S[1]**2+S[2]**2+S[3]**2+S[4]**2+S[5]**2)/(S[0]+S[1]+S[2]+S[3]+S[4]+S[5])))
```

```
def zero(n):
    Z=[]
    for i in range(n):
        Z.append(echelle2(alea(500)))
    return(min(Z))
```

```
image1= Image.open("empreinte 1.jpg")
empreinte1 = np.array(image1)
```

```
#Convertion des images:

def conversionnuancedegris(L):#Ici on prend en argument un tableau extrait d'une image RVB
    n=len(L)
    m=len(L[0])
    g=[]
    G=[]
    for i in range(n):
        for j in range(m):
            g.append(L[i][j][0]*0.299+L[i][j][1]*0.587+L[i][j][2]*0.114)
        G.append(g)
        g=[]
    return(G)

def conversirnoirblanc(L):#Ici on prend en argument un tableau converti en nuances de gris
    n=len(L)
    m=len(L[0])
    nb=[]
    NB=[]
    for i in range (n):
        for j in range(m):
            if L[i][j]<=200:
                nb.append(1)
            else:
                nb.append(0)
        NB.append(nb)
        nb=[]
    return(NB)

def convert(L):
    return(conversirnoirblanc(conversionnuancedegris(L)))
```