

# Contrôle du flux sanguin chez les malades atteints de problèmes vasculaires et prévention en cas de crise

**BENMANSOUR MOHAMMED – NUM SCEI 16822**

# PLAN :

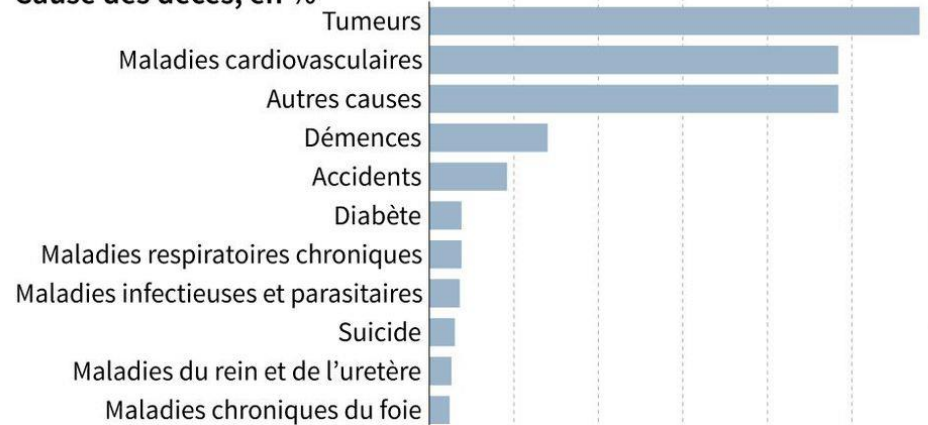
- Problématique
- Mesure du flux sanguin (Effet Doppler)
- Correction du flux sanguin
  - Propriétés rhéologiques du sang
  - Réponse des globules rouges à un champs magnétique extérieur
- Modélisation d'un appareil miniature
- Conclusion

# Problématique

## La mortalité en France

Chiffres 2016 pour la France métropolitaine

Cause des décès, en %



Causes de mortalité en France

# I - Mesure du flux sanguin

## Effet Doppler :

### L'exemple de l'ambulance



## Expression de la fréquence reçue

- $F_r$  : fréquence reçue
- $F_e$  : fréquence émise
- $V$  : vitesse de l'onde dans le milieu
- $V_o$  : vitesse de l'observateur
- $V_s$  : vitesse de la source

$$F_r = F_e \frac{V - V_o}{V - V_s}$$



# Echographie Doppler

- La sonde est inclinée d'un angle  $\theta$ .
- L'onde rencontre une hématie (globule rouge)

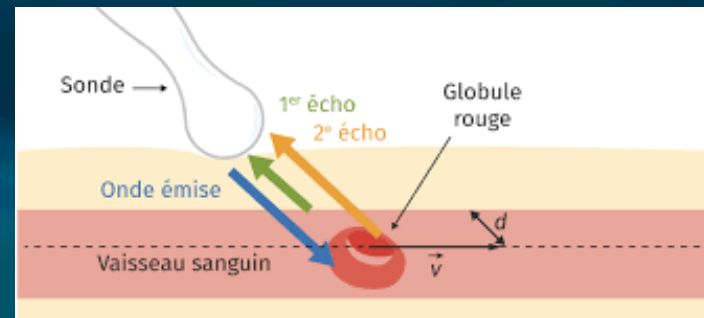


Schéma explicatif de l'effet Doppler

$$F_g = F_e \times \frac{C + V \cos \theta}{C}$$

↑  
Fréquence reçue par  
le globule rouge

$$F_r = F_g \times \frac{C}{C - V \cos \theta}$$

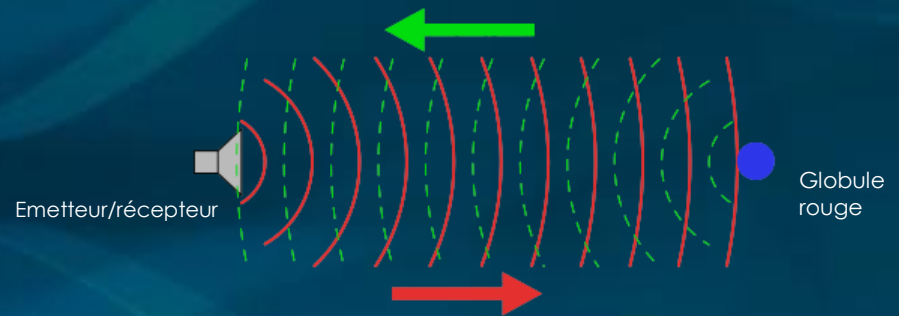


Schéma d'onde émise et réfléchi par une hématie



## Expression finale

$$F_e = F_r \frac{C + V \cos \theta}{C - V \cos \theta}$$

C : Célérité des ultrasons (1480 m/s)

V : Vitesse des hématies (du sang)

$\theta$  : Angle d'incidence

Cela donne :

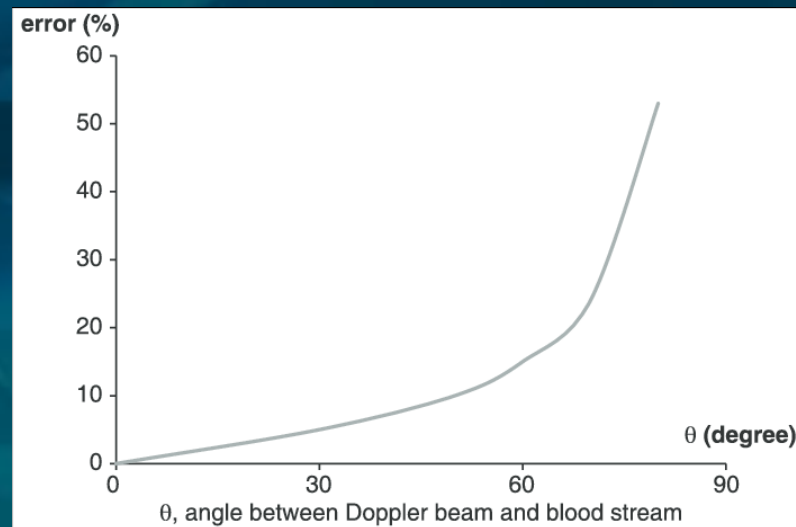
$$\Delta F = F_e - F_r = \frac{2FV \cos \theta}{c - V \cos \theta}$$

$$\begin{matrix} V \ll c \\ \Rightarrow \end{matrix} \quad \frac{\Delta F}{F} \approx \frac{2V \cos \theta}{c}$$

# Choix optimal de l'angle

La marge d'erreur augmente quand  $\cos\theta$  diminue

Des résultats expérimentaux confirment cela:



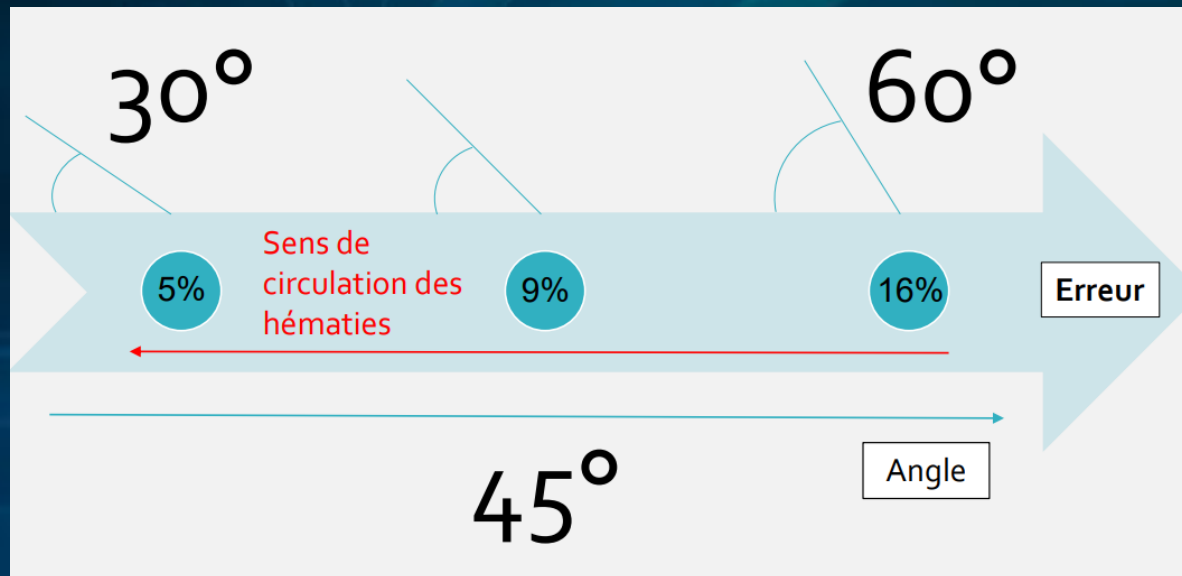
Erreur en % en fonction de l'angle  $\theta$

Les petites valeurs de l'angle  $\theta$  ne sont pas pratiques



Application réelle de l'échographie Doppler

Il vaut mieux donc éviter les angles limites :  $0^\circ$  et  $90^\circ$



Angles et marge d'erreur de l'échographie

## II - Correction du flux sanguin

## II – 1) Propriétés rhéologiques du sang



# Différents types de fluide

Fluide  
newtonien

Fluide rhéoépaississant

Fluide  
rhéofluidifiant



Eau liquide

$$\eta = \text{cte}$$

# Différents types de fluide

Fluide newtonien

Fluide  
rhéoépaississant

Fluide  
rhéofluidifiant



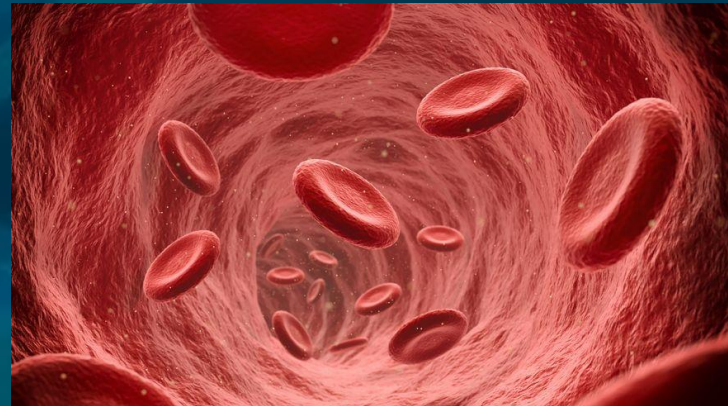
Suspensions concentrées de fécule de maïs

# Différents types de fluide

Fluide newtonien

Fluide rhéoépaississant

Fluide  
rhéofluidifiant

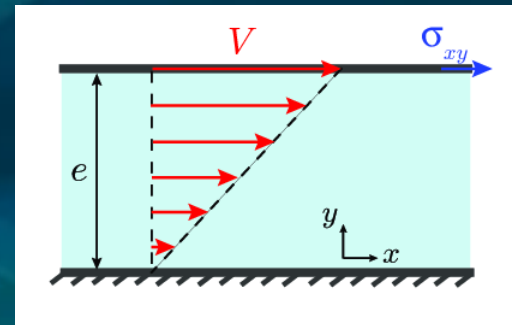


Sang dans les veines

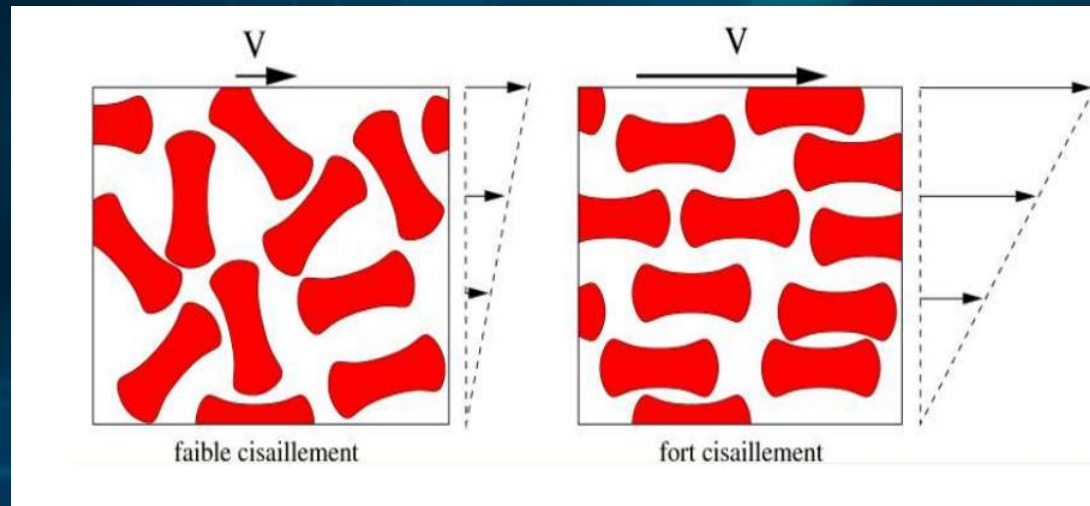
# Contrainte de cisaillement

C'est le gradient de vitesse entre les couches du fluide

$$\tau = \overrightarrow{grad} V$$



# Contrainte de cisaillement



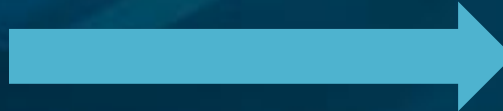


Alignement des globule rouges



Grande contrainte de cisaillement

*Propriétés rhéofluidifiantes du sang*



Faible viscosité

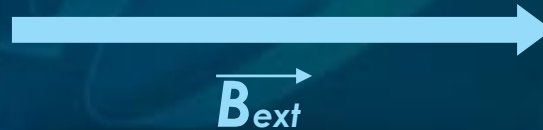


Plus grand flux sanguin

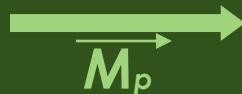
## II – 2) Réponse des globules rouges à un champ magnétique extérieur



# Les matériaux diamagnétiques et paramagnétiques



Matériau Paramagnétique



Matériau Diamagnétique

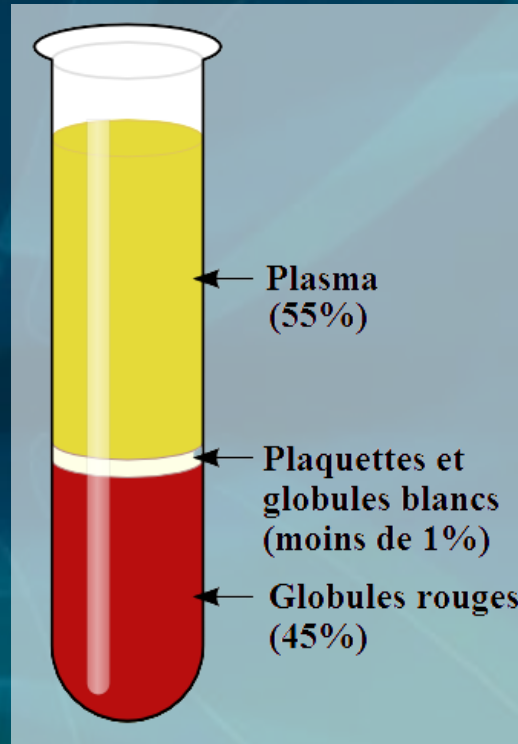


## Aimantation de tels matériaux

$$\overrightarrow{M_p} = \frac{\chi_p}{1 + \chi_p \cdot D} \overrightarrow{H_{ext}}$$

- $\overrightarrow{M_p}$  : Moment magnétique de l'objet
- $\chi_p$  : Susceptibilité magnétique (propre au matériau)
- $D$  : Constante de désaimantation
- $\overrightarrow{H_{ext}}$  : Champ d'excitation magnétique extérieur, défini par :  $\overrightarrow{H_{ext}} = \frac{\overrightarrow{B_{ext}}}{\mu_0 \mu_r}$

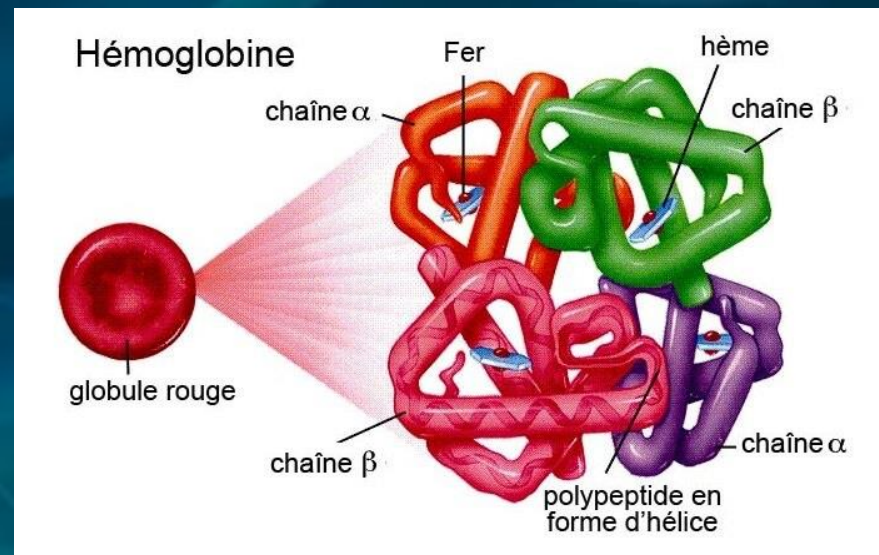
# Composition du sang



Composition du sang

# Composition d'un globule rouge

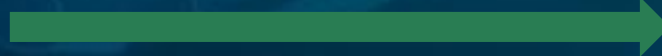
- Un globule rouge est constitué d'eau et d'hémoglobine



Composition d'un globule rouge

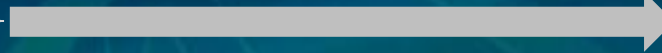


**Diamagnétique**

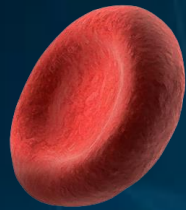


**Un peu paramagnétique**

+



**Non paramagnétique**



$$\chi_{gr} \sim -6.10^{-6}$$

Comportement diamagnétique



$$\chi_{gr} \sim -9.10^{-6}$$

Donc finalement l'expression du moment magnétique créé est :

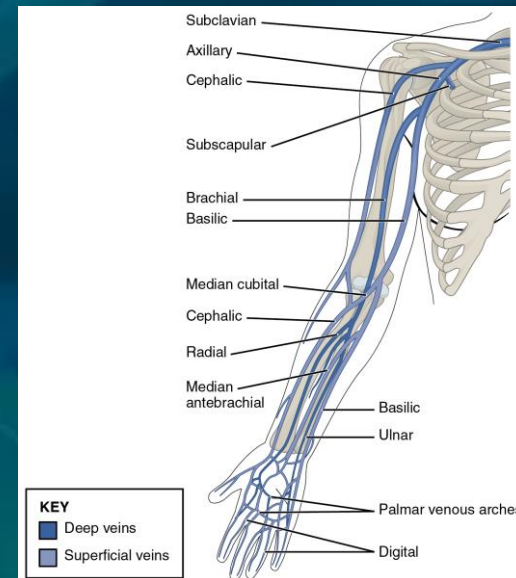
$$\overrightarrow{M_{gr}} \approx \chi_{gr} \cdot \frac{\overrightarrow{B_{ext}}}{\mu_0 \mu_r}$$



# Frottement fluide dans le sang

L'écoulement majoritaire dans le bras humain est un écoulement laminaire

$$Re \ll 1$$

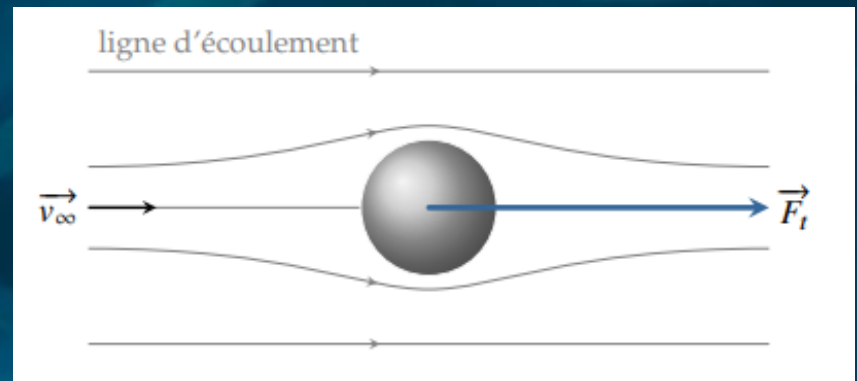


Système veineux des bras humains

La force de frottement de fluide pour les sphères de petite dimension est donnée par la formule de stockes :

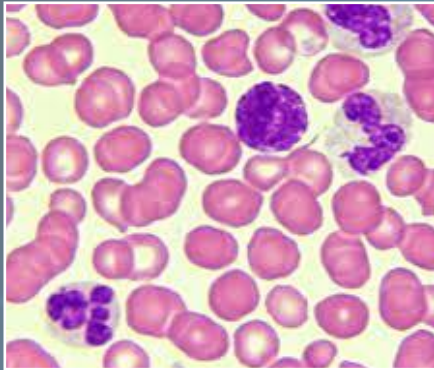
$$\vec{F}_t = -6\pi\eta r\vec{v}$$

- $\eta$  : Viscosité du fluide
- $r$  : Rayon de la sphère
- $\vec{v}$  : Vitesse de la sphère par rapport au fluide



Sphère dans un fluide en écoulement laminaire

On peut assimiler les globules rouges à des sphères dans le sang, de rayon  $r = 4.10^{-6}$  mm, baignant dans du plasma en mouvement, qui est toujours en écoulement laminaire.



Sang sous microscope

Donc dans un repère associé au plasma en mouvement :

$$\eta_{plasma} = cte \approx 1.3cP$$

$$\vec{F}_t \approx -0.0001 \times \vec{v}$$

## Contrainte sur le temps d'application du champ

La Vitesse du sang dans les veines des bras est d'environ 25cm/s (pour des anomalies 20cm/s)  
Donc le sang traversera la zone soumise au champ en environ 1s

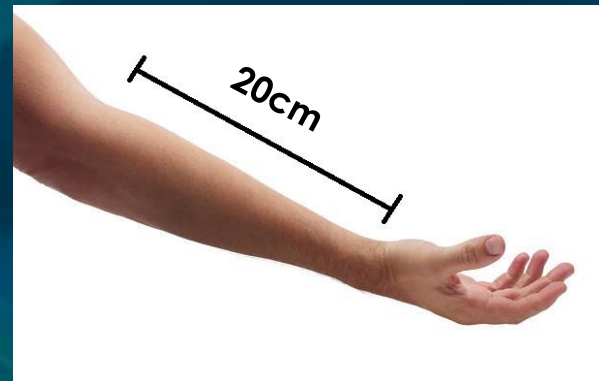
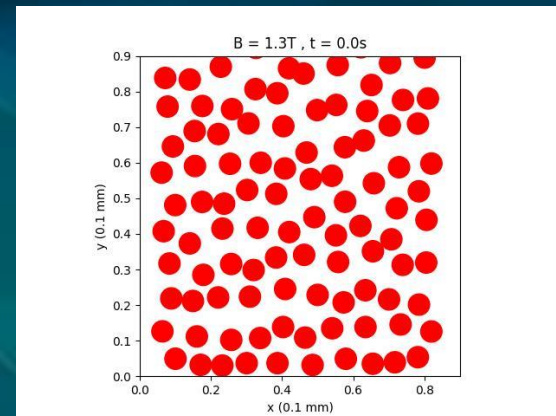
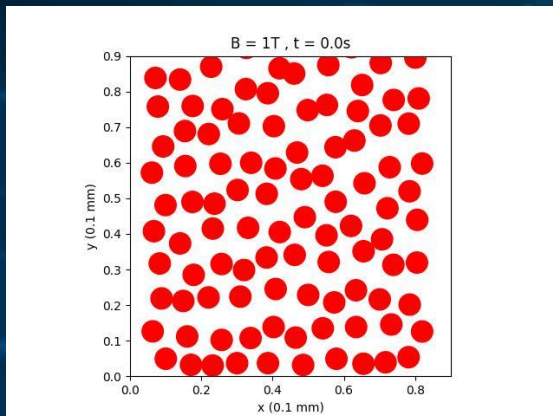
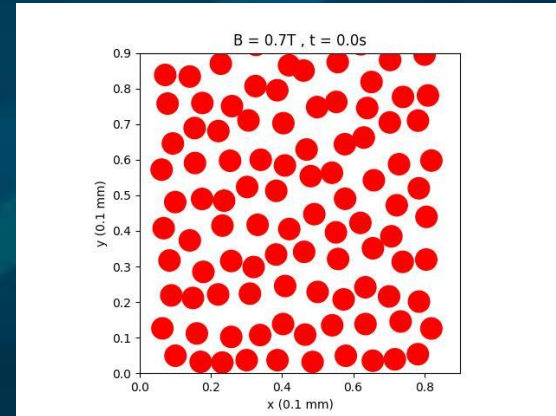
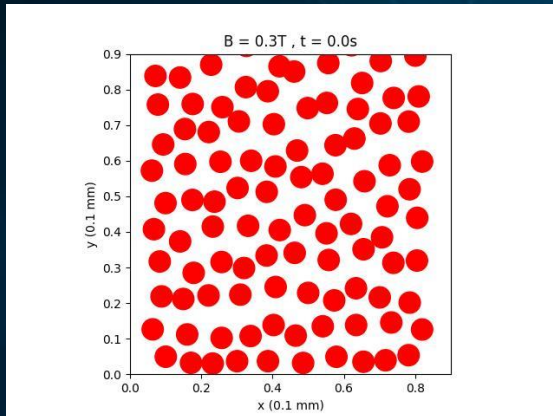


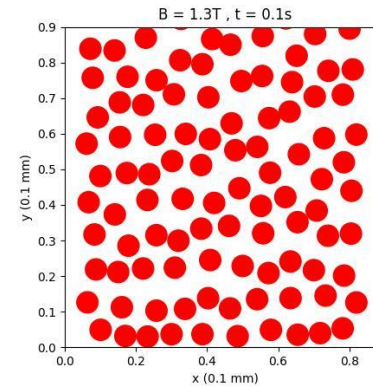
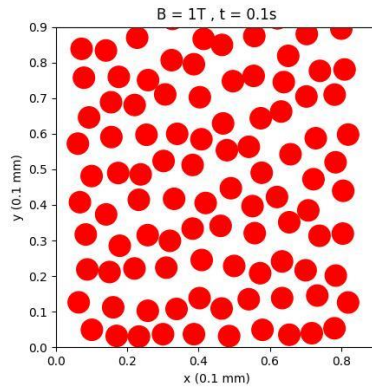
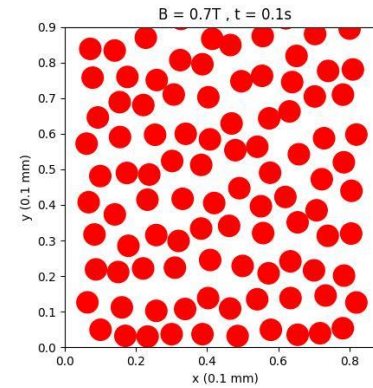
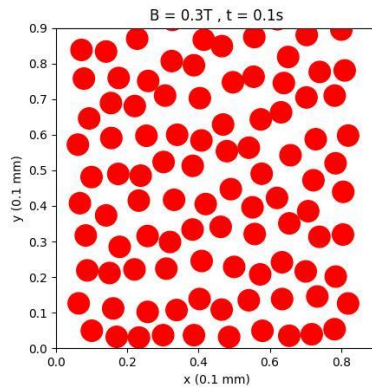
Schéma du bras humain

# Simulation du comportement des globules rouges

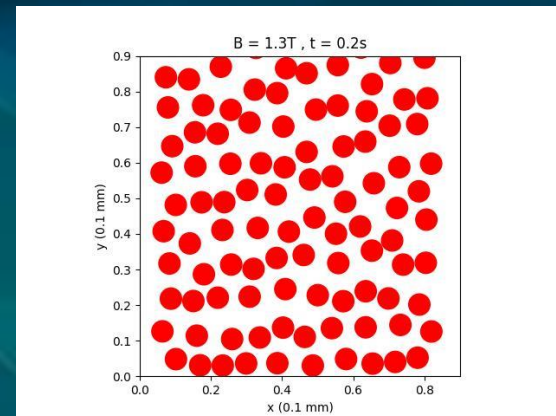
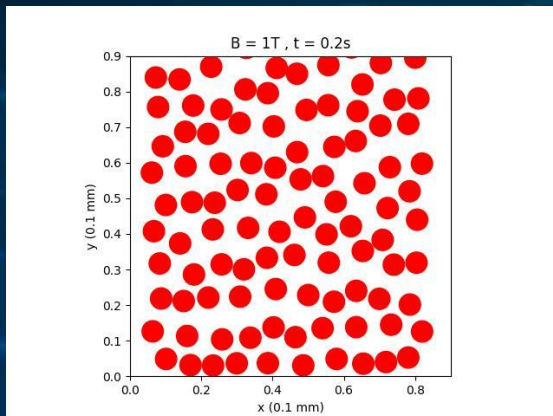
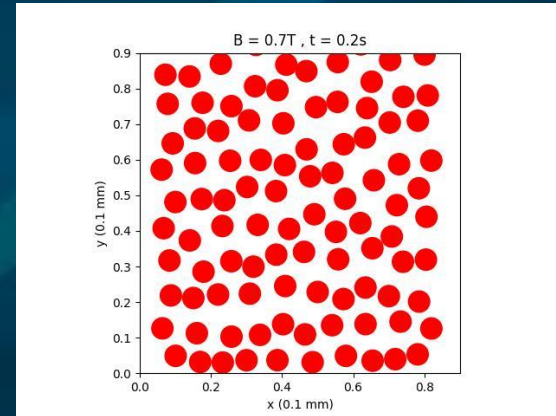
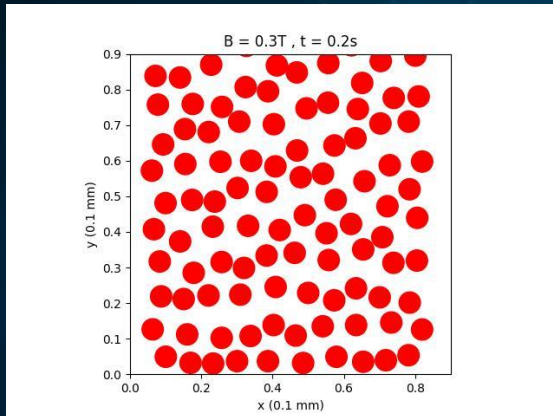




# Simulation du comportement des globules rouges

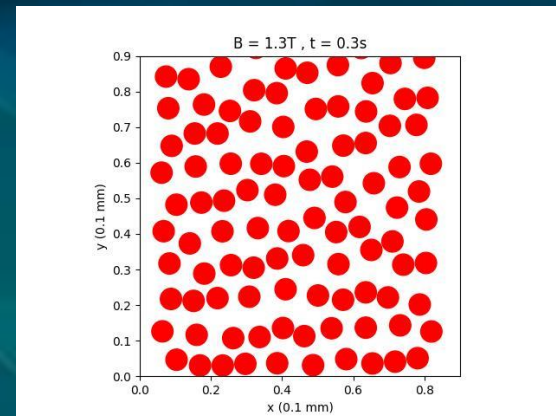
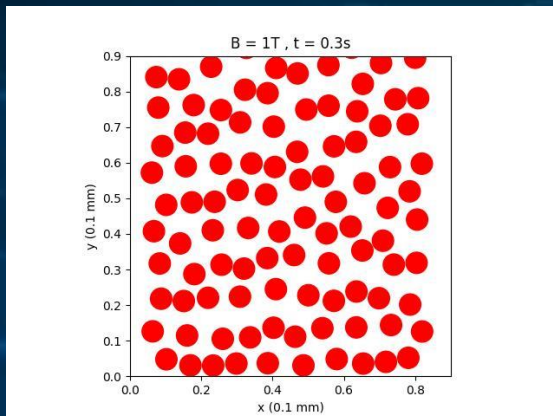
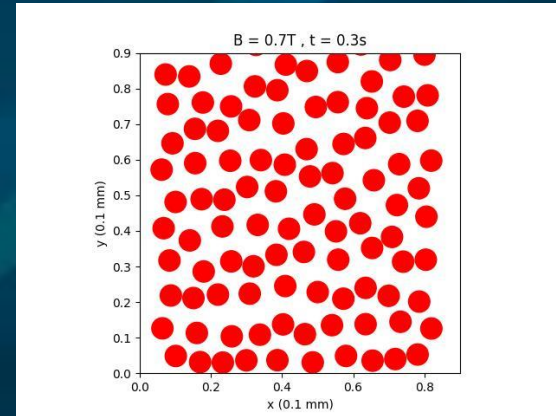
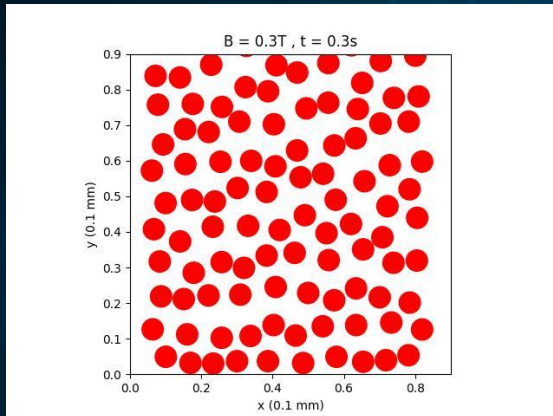


# Simulation du comportement des globules rouges

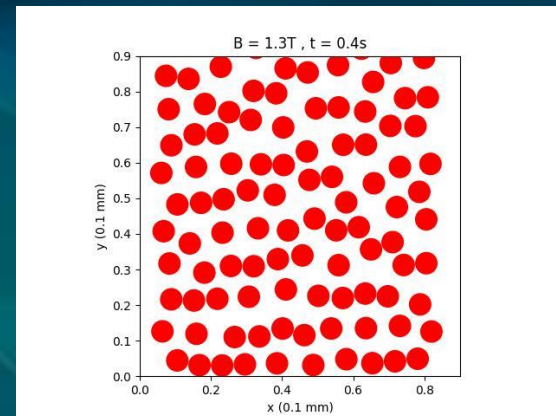
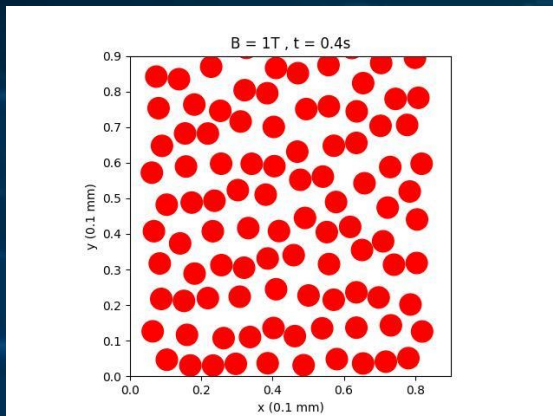
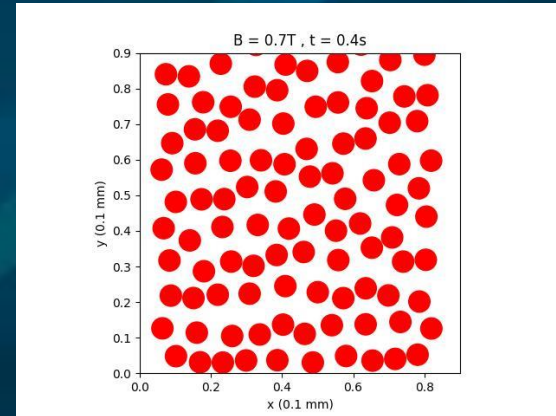
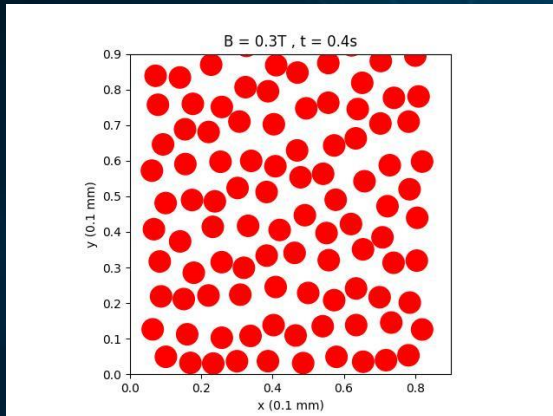




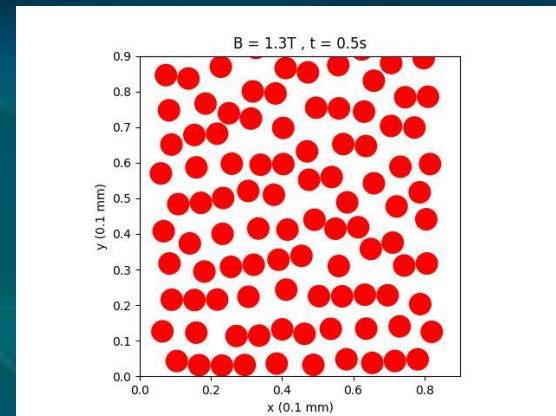
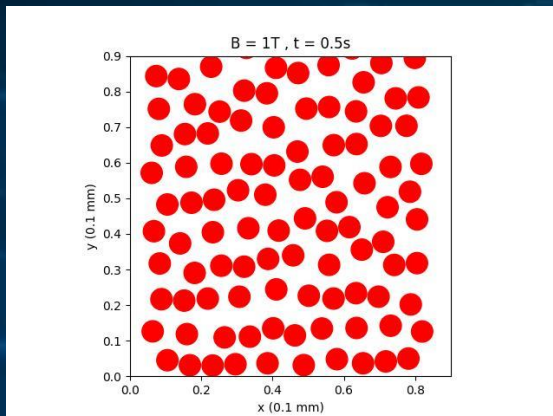
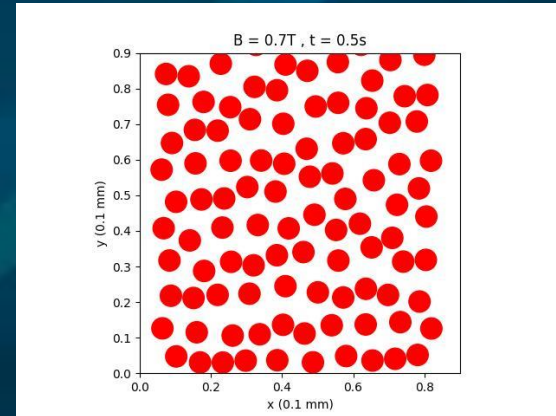
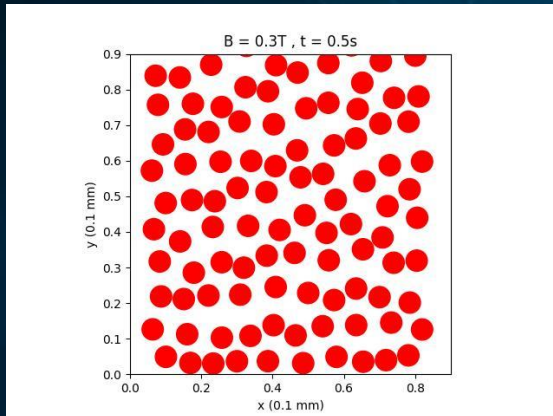
# Simulation du comportement des globules rouges



# Simulation du comportement des globules rouges

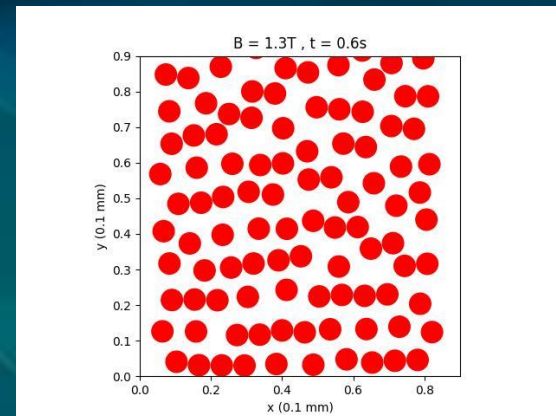
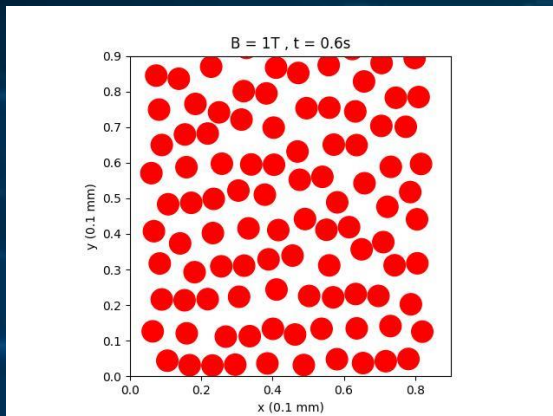
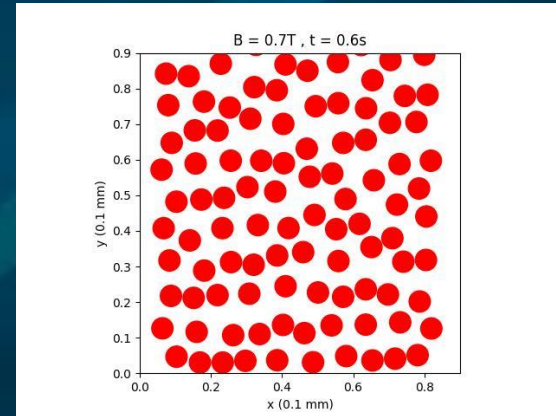
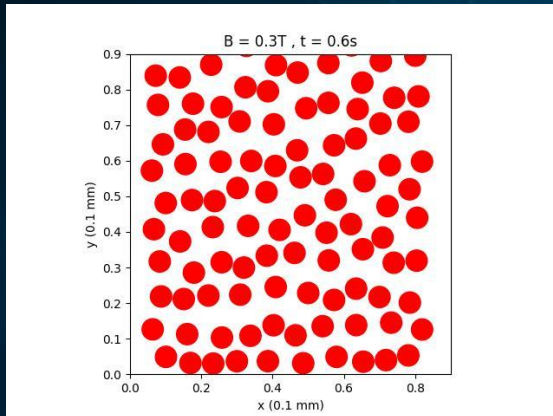


# Simulation du comportement des globules rouges

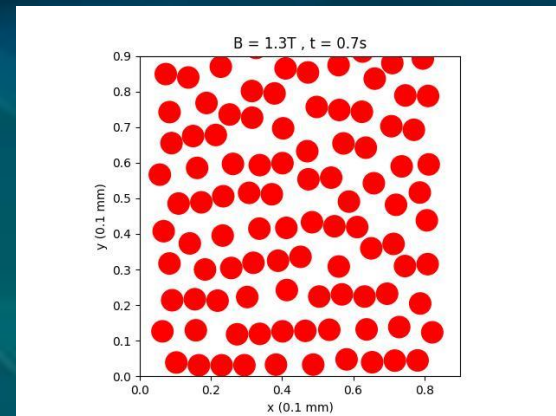
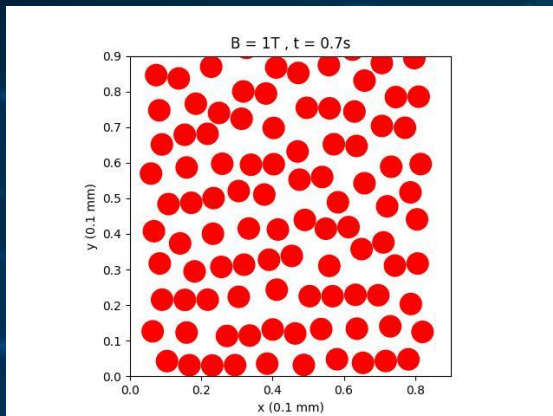
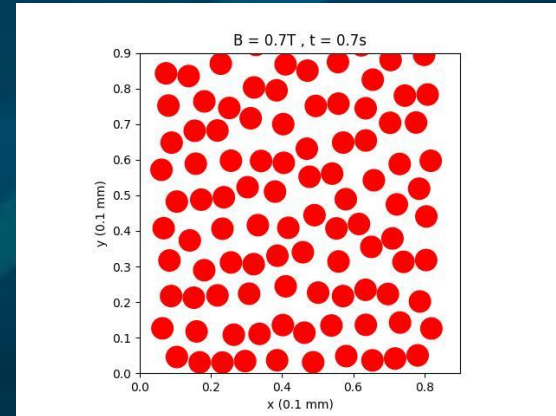
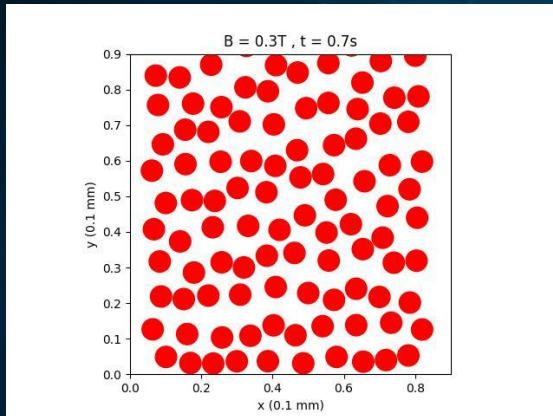




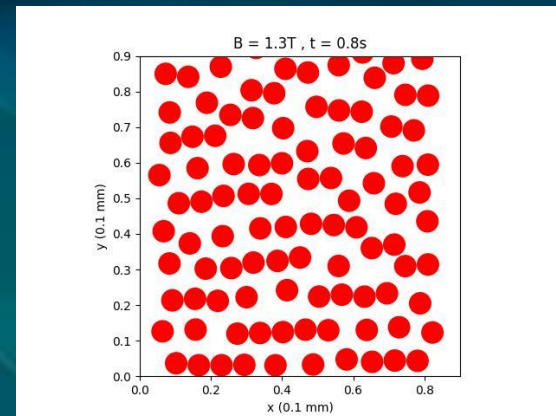
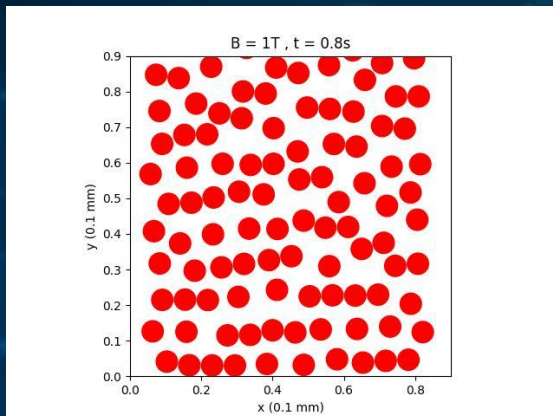
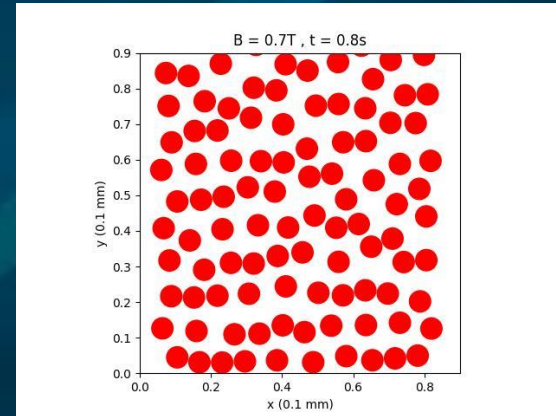
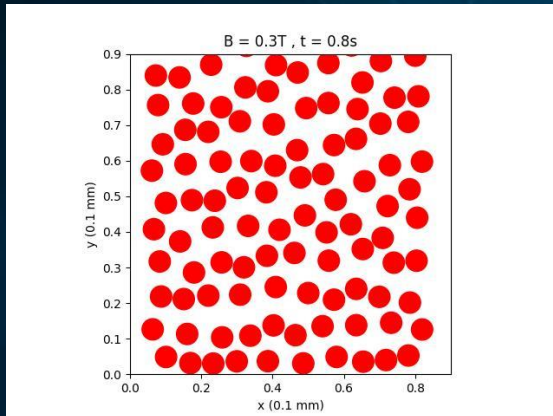
# Simulation du comportement des globules rouges



# Simulation du comportement des globules rouges

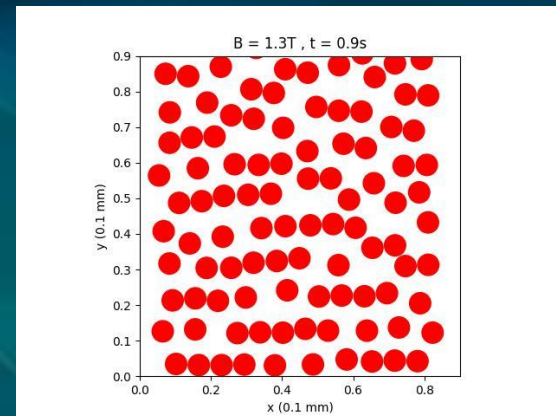
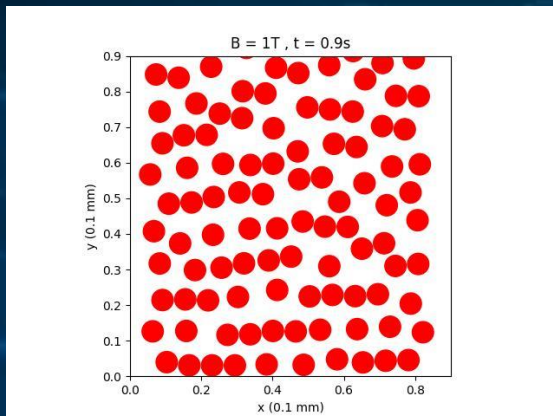
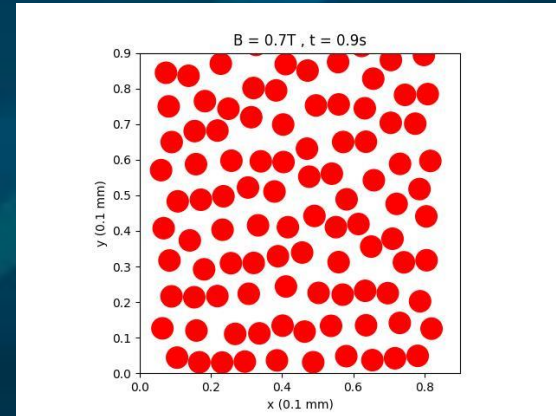
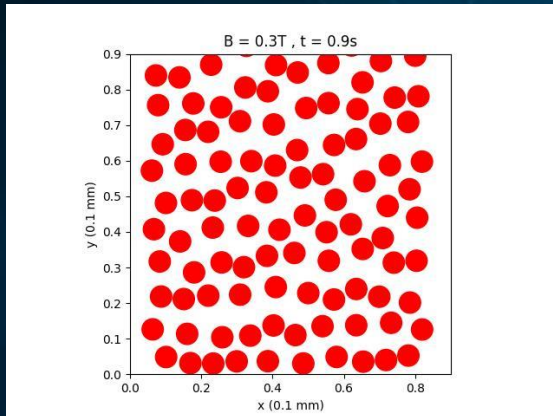


# Simulation du comportement des globules rouges

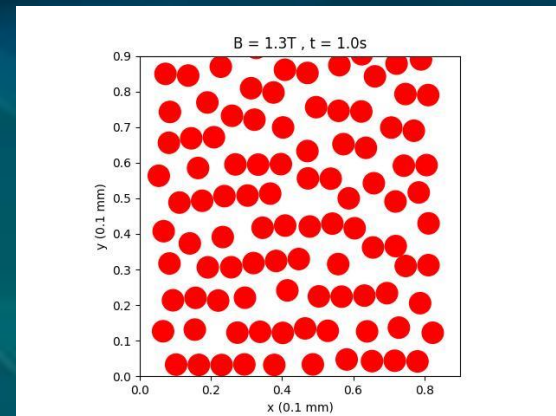
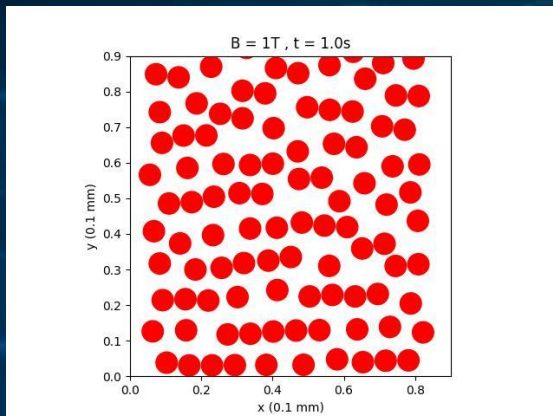
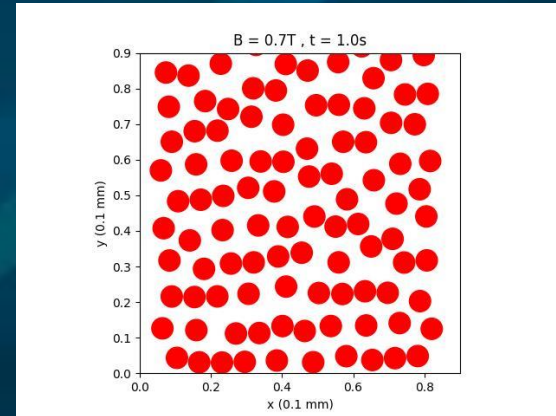
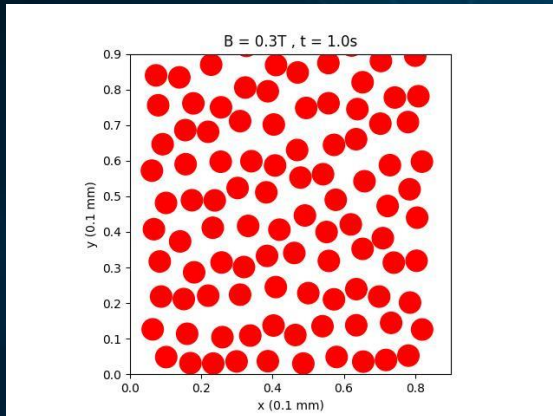




# Simulation du comportement des globules rouges



# Simulation du comportement des globules rouges



Donc pour un champ magnétique  $\vec{B}$  supposé uniforme le long de l'avant-bras :

- La valeur minimale de B : 1 Tesla
- La valeur optimale de B : 1.3 Tesla

# III - Modélisation d'un appareil miniature

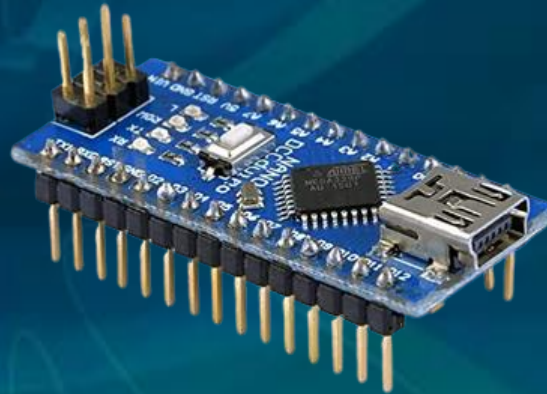
# Cahier de charges

Il faut un appareil qui puisse :

- Émettre et recevoir des ultrasons
- Mesurer la différence de fréquence
- Précision de cette mesure
- Créer un champ magnétique uniforme le long de l'avant-bras
- Ce champ doit être de l'ordre de 1 Tesla
- Ce champ doit être parallèle au bras
- Avoir une carte électronique pour tout raccorder



# La carte électronique (pour prototype)



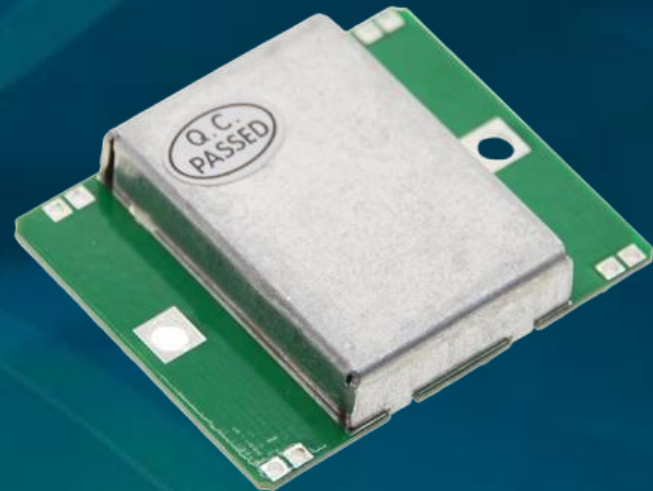
Arduino Nano



# Emetteur/Capteur ultrasons

HB100 est un capteur à effet Doppler

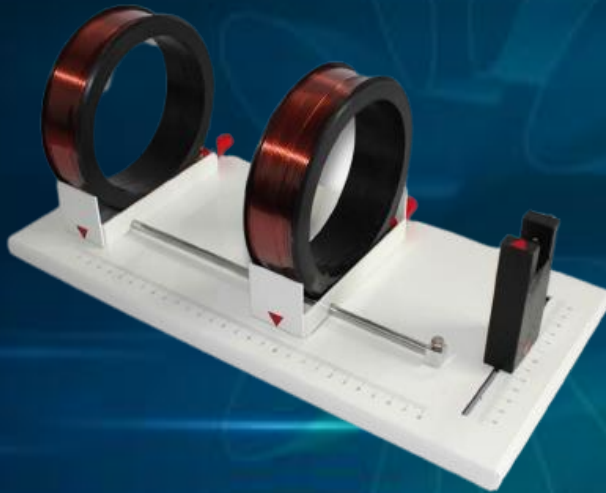
Facilement programmable avec l'arduino



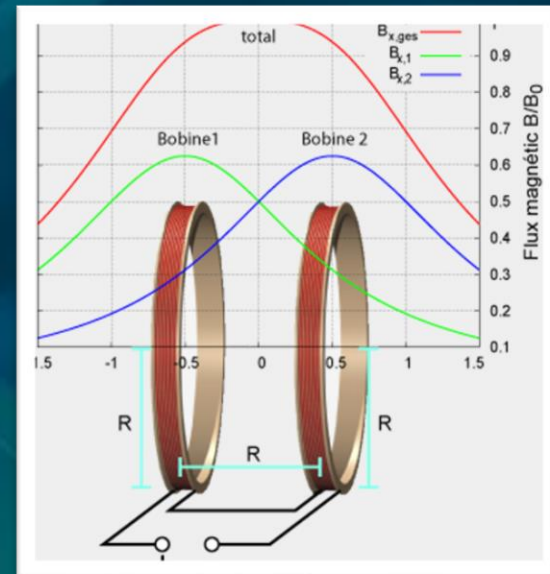
Capteur HB100

# Créer un champ magnétique uniforme

## Les bobines de Helmholtz



Bobines de Helmholtz

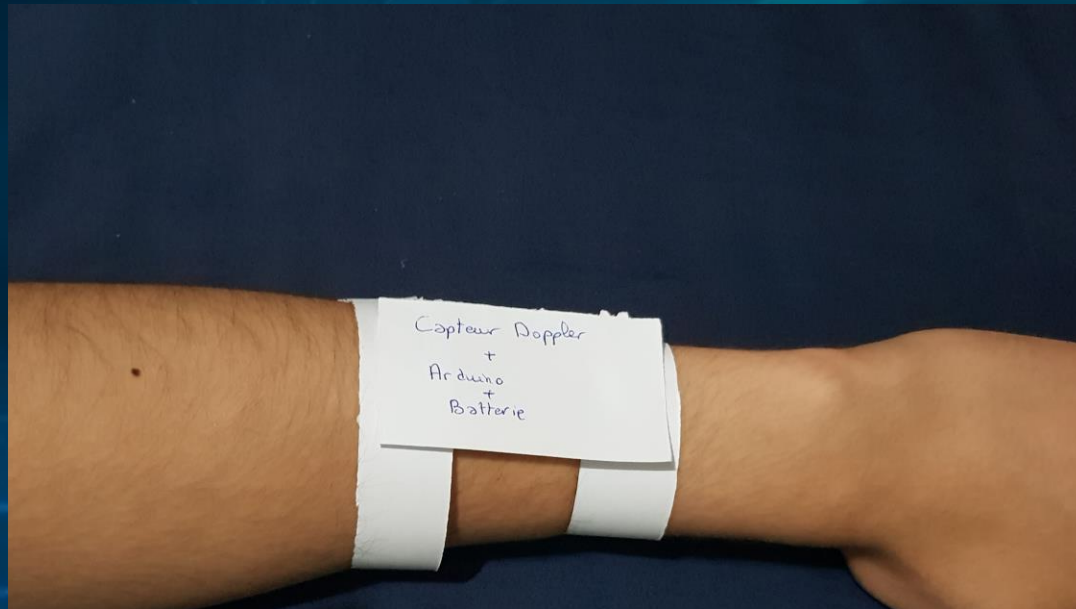


Flux magnétique généré par les bobines de Helmholtz



Idée de placement des bobines de Helmholtz

# Schéma final de l'appareil

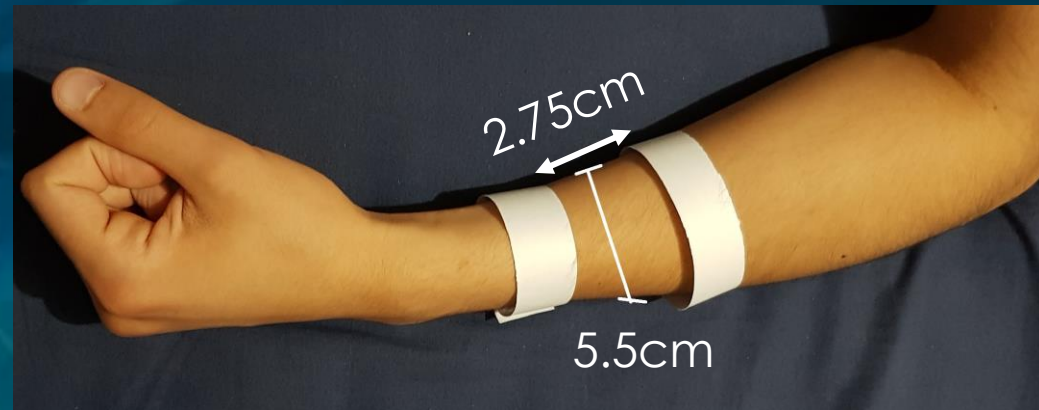


Idée de conception de l'appareil

# Limite de ce modèle

## Étendue du champ magnétique

Le champ uniforme ne s'étendra pas plus de 3cm  $\ll$  20cm (longueur de l'avant-bras)



Mise à l'échelle de l'appareil



## Intensité du champ magnétique




Générateur de courant et bobines d'Helmholtz



Teslamètre donnant la valeur du champ créé par les bobines



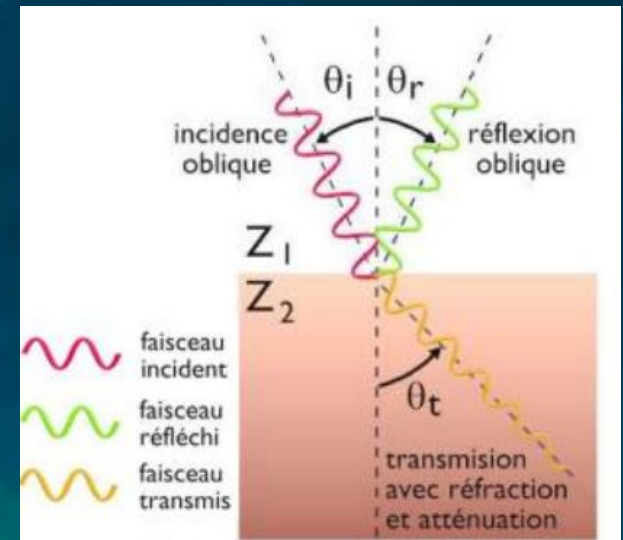


Il faudrait au moins 550A pour créer  
un tel champ, assez pour détruire les  
bobines

## Perturbations et réflexions de l'onde sonore



Echographie Doppler avec gel



Réflexion de l'onde sonore sur la peau

# Conclusion

# Cahier de charges

Compte rendu des exigences respectées:

- Émettre et recevoir des ultrasons
- Mesurer la différence de fréquence
- Précision de cette mesure
- Créer un champ magnétique uniforme le long de l'avant-bras
- Ce champ doit être de l'ordre de 1 Tesla
- Ce champ doit être parallèle au bras
- Avoir une carte électronique pour tout raccorder

A stylized, translucent plant with long, pointed leaves is centered in the background. The leaves are a light blue-green color, contrasting with the dark blue background. The plant appears to be a type of succulent or a stylized flower.

# Merci pour votre attention

# Annexe



# Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
1 import matplotlib.pyplot as plt
2 import matplotlib.animation as anim
3 import numpy as np
4 import math
5 import random
6 import os
7 |
8
9 print("Calculating..")
10 a = 0 # Positive value => Same Charges attract each other, like gravitation / Negative value =>
11 | # Electrostatique
12 rep = 0.1
13 dynamiccollision = False
14 collision_factor = 0
15 g = 0 # Gravity
16 f,f_e= 1.5,1 # Coefficient de frottement + exposant de frottement
17 random_deplac = 0
18 random_decoy = 30
19 random_proba = 5 # value in percent
20 magneticMu = 65.91 * (10**(3))
21
22 B = 1.3
23
24 NBall = 100
25 tf = 1.1
26 n = int(tf*180)
27 rayon = .03
28 fieldForce = 0.002
29 fps = 30
30 fpsim = 10
31 step= int(n/(tf*fps))
32 stepim= int(n/(tf*fpsim))
33 outputType = 2 # 0 for video, 1 for frames, 2 for both (for video, you will need ffmpeg
34 outputName = 'B-1.3-Tes'
35 title = 'B = 1.3T'
36
37 maxFieldLength = 0.9
38
```

# Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
37 maxFieldLength = 0.9
38
39 N_chunk = 40
40 chunks = [[[] for j in range(N_chunk)] for i in range(N_chunk)]
41 nearByLength = 4
42
43
44
45
46 def getNearBy(i,X,j):
47     nearBalls = []
48     x,y = (X[i][j-1,0]*N_chunk)//maxFieldLength,(X[i][j-1,1]*N_chunk)//maxFieldLength
49     x,y = int(x),int(y)
50     if x >= N_chunk: x = N_chunk-1
51     if y >= N_chunk: y = N_chunk-1
52     if x < 0: x = 0
53     if y < 0: y = 0
54     for k in range(-nearByLength,nearByLength+1):
55         if k+x<0 or k+x>=N_chunk:
56             continue
57         for l in range(-(nearByLength-abs(k)),nearByLength+1-abs(k)):
58             if l+y<0 or l+y>=N_chunk:
59                 continue
60             nearBalls.extend(chunks[x+k][y+l])
61     return nearBalls
62
63
64 def N2(u):
65     return math.sqrt(u[0]**2+u[1]**2)
66 def ps(u,v):
67     return (u[0]*v[0]+u[1]*v[1])
68 def orth_proj(u,v):
69     return (ps(u,v)/ps(v,v))*v
70 rayon2 = rayon +fieldForce
71 def setPos(i):
72     #P=[(5*k+10,94-5*l) for k in range(15) for l in range(15)]
73     P= []
```

## Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
74     for k in range(10):
75         start=rayon+(1+math.cos(k+math.exp(k)))/80
76         for l in range(10):
77             P.append((0.08+0.08*k+(math.cos(k+10*l)/50),start))
78             start += (1+math.cos(k*l+math.exp(1-k+10)))/40 + 0.07
79     if i<len(P):
80         return np.array( P[i])
81     return np.array([random.randint(1,maxFieldLength-1),random.randint(1,maxFieldLength-1)])
82 def setVelocity(i):
83     Vel=[]
84     if i<len(Vel):
85         return np.array( Vel[i])
86     return np.zeros(2)
87 def setMass(i,j=0):
88     Mass=[]
89     if i<len(Mass):
90         return Mass[i]
91     return 0.7
92 def setCharge(i):
93     Attr = [1,1]
94     if i<len(Attr):
95         return Attr[i]
96     return 1
97 def setMagnet(i,j=0):
98     return 9*(10**(-6))*B
99 def setColor(i):
100    return 'red'
101    Col = ['red']
102    if i>=len(Col):
103        return ''
104    return Col[i]
105
106 def forces(i,X,V,A,j=1):
107     attraction = np.array([0.,0.])
108     magneticForce = np.array([0.,0.])
109     velocity_action = V[i][j-1]
110     velocity_correction = np.array([0.,0.])
```

## Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
110 velocity_correction = np.array([0,0,0])
111 repulsion = [0,0]
112 frott = [0,0]
113 if fl!=0:
114     vit = N2(V[i][j-1])
115     if vit!=0:
116         frott = -f*(vit**(f_e-1))*V[i][j-1]
117 for k in getNearBy(i,X,j):
118     if i!=k:
119         dist = N2(X[i][j-1]-X[k][j-1])
120         coeff = setCharge(k)*setCharge(i)
121         if dist==0:
122             continue
123         attraction += -a*coeff/dist*(X[i][j-1]-X[k][j-1])
124         if magneticMu!=0 and setMagnet(i,j)!=0 and setMagnet(k,j)!=0 and dist!=0:
125             rel_pos = X[i][j-1]-X[k][j-1]
126             #rel_pos = np.array([0,-1])
127             cos = ps(rel_pos,(1,0))/dist
128             if rel_pos[1]>=0:
129                 teta = math.acos(cos)
130             if rel_pos[1]<0:
131                 teta = -math.acos(cos)
132             teta += math.pi/2
133             er = np.array([math.sin(teta),-math.cos(teta)])
134             eteta = np.array([math.cos(teta),math.sin(teta)])
135             basicME= magneticMu*setMagnet(i,j)*setMagnet(k,j)*1/(2*math.pi*(dist**4))
136             magneticForce += 3*basicME*((math.cos(teta)*math.sin(teta)))*eteta
137             if not dist<=2*rayon2:
138                 magneticForce += basicME*3*(-0.5*(math.sin(teta)**2)+(math.cos(teta)**2))*er
139             #print(magneticForce)
140             #print(teta*180/math.pi)
141         if dist<=2*rayon2:
142             if dynamicCollision:
143                 repulsion += ((2*rayon2-dist)**2)*rep/(dist)*(X[i][j-1]-X[k][j-1])
144                 velocity_action += collision_factor*X[i][j-1]
145             else:
146                 mi,mk = setMass(i,j),setMass(i,k)
147                 avg_velo = (1/(mi+mk))*(mi*velocity_correction + mk*V[k][j-1])
```

# Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
148     vel_in = V[i][j-1] - avg_velo
149     pos_in = X[i][j-1] - X[k][j-1]
150     vel_proj = orth_proj(vel_in, pos_in)
151     vel_dist = N2(vel_proj)
152     velocity_action = avg_velo + vel_in - vel_proj + ((2*rayon2-dist)/rayon2 + vel_dist + 0.01) * (collision_factor + 0.1) * 1/dist * pos_in
153     gravity = setMass(i, j) * g * np.array([0, -1])
154
155     return [attraction + repulsion + frott + gravity + magneticForce, velocity_action]
156
157
158
159
160
161 M = []
162 X = []
163 V = []
164 A = []
165 for i in range(NBall):
166     X.append(np.zeros((n, 2)))
167     V.append(np.zeros((n, 2)))
168     A.append(np.zeros((n, 2)))
169     X[i][0] = setPos(i)
170     V[i][0] = setVelocity(i)
171     M.append(setMass(i))
172
173 for j in range(1, n):
174     # Initialize chunks
175     chunks = [[[] for j in range(N_chunk)] for i in range(N_chunk)]
176     for i in range(NBall):
177         x, y = (X[i][j-1, 0] * N_chunk) // maxFieldLength, (X[i][j-1, 1] * N_chunk) // maxFieldLength
178         x, y = int(x), int(y)
179         if x >= N_chunk: x = N_chunk - 1
180         if y >= N_chunk: y = N_chunk - 1
181         if x < 0: x = 0
182         if y < 0: y = 0
183         chunks[x][y].append(i)
184
```



## Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
185
186     for i in range(NBall):
187         resultant = forces(i,X,V,A,j)
188         A[i][j] = (1/M[i]) * resultant[0]
189         wonsk=0
190         if j*tf/n<random_decoy and random.randrange(0,100)<random_proba:
191             teta = random.randrange(0,360)*math.pi/180
192             wonsk = random_deplac*np.array([math.cos(teta),math.sin(teta)])
193         V[i][j] = resultant[1] + A[i][j]*tf/n + wonsk
194         X[i][j] = X[i][j-1] + resultant[1]*tf/n + 0.5*A[i][j-1]*((tf/n)**2)*0
195         if (X[i][j,0]>maxFieldLength-rayon or X[i][j,0]<=-rayon):
196             if V[i][j,0] * (maxFieldLength/2-X[i][j,0]) <0:
197                 V[i][j,0] *= -1
198         if (X[i][j,1]>maxFieldLength-rayon or X[i][j,1]<=-rayon):
199             if V[i][j,1] * (maxFieldLength/2-X[i][j,1]) <0:
200                 V[i][j,1] *= -1
201
202
203     fig = plt.figure()
204     axes = plt.axes()
205     axes.set_xlim((0,maxFieldLength))
206     axes.set_ylim((0,maxFieldLength))
207     axes.set_aspect(1)
208     axes.set_xlabel('x (0.1 mm)')
209     axes.set_ylabel('y (0.1 mm)')
210     axes.set_title(title)
211     circles = []
212
213
214     def init_anim():
215         for i in range(NBall):
216             if setColor(i)!='':
217                 circle=(plt.Circle((0,0),rayon,color=setColor(i)))
218             else:
219                 circle=(plt.Circle((0,0),rayon))
220             circles.append(circle)
221             axes.add_artist(circle)
222
```

## Code de la simulation des globules rouges

```
SimulationGlobulesRouges.py
"""
223 def render(j,step=step):
224     axes.set_title(title+" , t = "+str(int(tf*j*step/n*10)/10)+'s')
225     for k in range(NBall):
226         circles[k].center = X[k][j*step,0],X[k][j*step,1]
227     return axes
228
229 print("Start Baking")
230
231 path=''
232 path=os.path.join(os.getcwd(),outputName)
233 if not os.path.isdir(path):
234     os.mkdir(path)
235 init_anim()
236 for i in range(n//stepim):
237     render(i,stepim)
238     plt.savefig(os.path.join(path,outputName+'-'+str(i)+'.jpg'))
239
240 print("Finished")
241
```