IMPACT DES MESURES DE DISTANCIATION SOCIALE DANS LA PRÉVENTION DU COVID 19

Thème: Enjeux sociétaux: Environnement, Energie, Sécurité

Numéro d'inscription: 42655

PLAN:

1. **Introduction**
2. **Modèle mathématique**
   a) Etablissement d'un modèle épidémiologique compartimental capable de décrire la progression de l'épidémie
   b) Mise en équation de l'épidémie: Détermination des constantes et grandeurs caractéristiques dans un cas réel: L'Italie
   c) Modélisation des mesures de distanciation sociale dans le cadre du modèle adopté et interprétations
3. **Modèle graphique / Simulation**
   a) Hypothèses du modèle
   b) Détails du modèle et implémentation
   c) Modélisation et implémentation des mesures de distanciation sociale
   d) Expérimentation et interprétations
4. **Annexe**

1) INTRODUCTION:

- L'épidémie de COVID 19, apparue en décembre 2019 dans la ville de Wuhan en Chine, s'est transformée en l'espace de quelques mois en une pandémie d'ampleur mondiale.
- Devant cette crise sanitaire globale, les systèmes de santé mondiaux se sont retrouvés à genoux devant l'immense vague de malade, ce qui fut exaspéré par la confusion et la maladresse du corps décisionnel.
- Les mesures de distanciation sociale en particulier ont longtemps été un point de controverse : bien que de nombreux spécialistes les considèrent essentielles au freinage de la propagation du COVID 19, les politiciens demeurent hésitant à les mettre en place étant donné leurs répercussions sur le secteur économique.
- Il serait alors intéressant de créer un modèle capable de décrire l'effet des mesures de distanciation sociale afin de quantifier leur impact sur l'épidémie et ainsi juger quant à leur efficacité.

2) MODÈLE MATHÉMATIQUE:

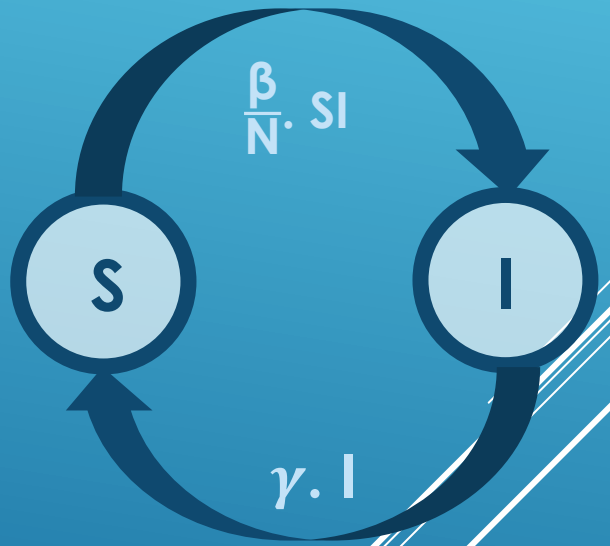
a) Etablissement d'un modèle épidémiologique compartimental capable de décrire la progression de l'épidémie

- Il existe un nombre de modèles mathématiques compartimentaux modélisant les épidémies, le plus simple étant le modèle SIS. Il est caractérisé par les équations:

$$\begin{cases} \dfrac{dS}{dt} = -\dfrac{\beta}{N}.SI + \gamma.I \\ \dfrac{dI}{dt} = \dfrac{\beta}{N}.SI - \gamma.I \end{cases}$$

S: Susceptibles
I: Infectés
β: Taux d'infection
γ: Taux de guérison

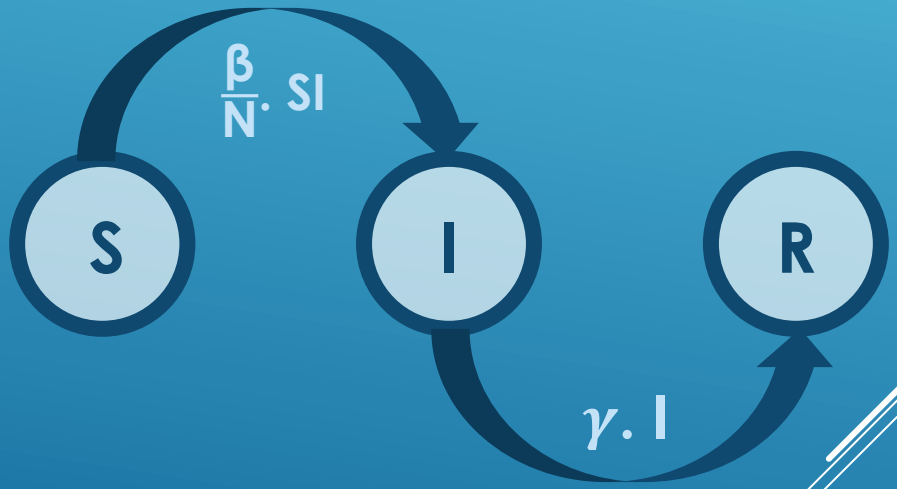
$$\frac{dS}{dt} + \frac{dI}{dt} = 0 \Rightarrow S + I = N = Constante$$



MODÈLE SIR

$$\begin{cases} \frac{dS}{dt} = -\frac{\beta}{N} . SI \\ \frac{dI}{dt} = \frac{\beta}{N} . SI - \gamma . I \\ \frac{dR}{dt} = \gamma . I \end{cases}$$

$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = 0 \Rightarrow S + I + R = N = Constante$$



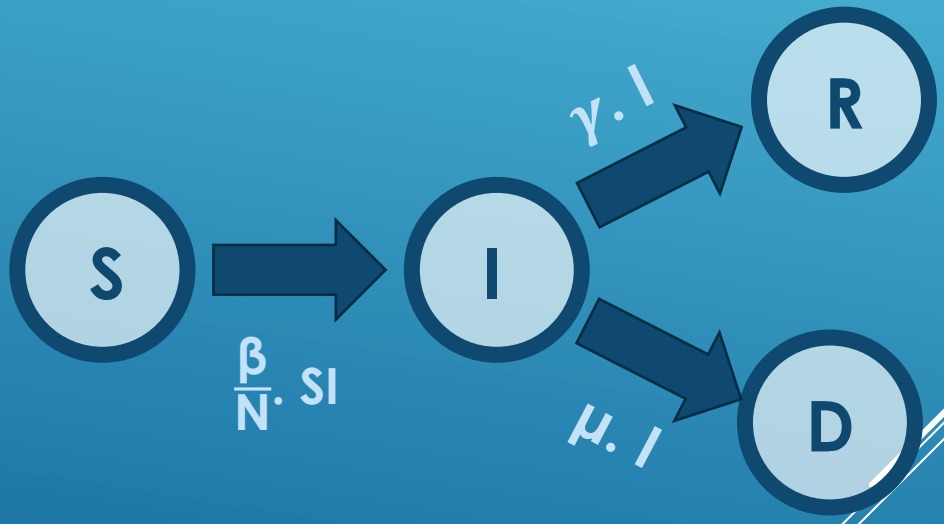
- **S:** Susceptibles
- **I:** Infectés
- **R:** Retirés
- **β:** Taux d'infection
- **γ:** Taux de guérison

MODÈLE SIRD

$$\begin{cases} \frac{dS}{dt} = -\frac{\beta}{N}.SI \\ \frac{dI}{dt} = \frac{\beta}{N}.SI - \gamma.I - \mu.I \\ \frac{dR}{dt} = \gamma.I \\ \frac{dD}{dt} = \mu.I \end{cases}$$

$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} + \frac{dD}{dt} = 0 \Rightarrow S + I + R + D = N = Constante$$

S: Susceptibles
I: Infectés
R: Rétablis
D: Décédés
β: Taux d'infection
γ: Taux de guérison
μ: Taux de décès



b) Mise en équation de l'épidémie: Détermination des constantes et grandeurs caractéristiques dans un cas réel: L'Italie (Méthode 1)

$$\begin{cases} \dfrac{dS}{dt} = -\dfrac{\beta}{N}.SI \\ \dfrac{dI}{dt} = \dfrac{\beta}{N}.SI - \gamma.I - \mu.I \\ \dfrac{dR}{dt} = \gamma.I \\ \dfrac{dD}{dt} = \mu.I \end{cases}$$

$$\begin{cases} \gamma = \dfrac{\frac{dR}{dt}}{I} \\ \mu = \dfrac{\frac{dD}{dt}}{I} \\ \beta = \dfrac{-N.\frac{dS}{dt}}{SI} \end{cases}$$

Régression polynomiale:

Méthode des moindres carrés

1) DÉFINITION: RÉGRESSION POLYNOMIALE

- La régression polynomiale est une approche statistique qui est employée pour modéliser une relation de forme non-linéaire entre la réponse (y) et la ou les variables explicatives (x)
- Pour prendre en charge cette forme non-linéaire de la relation entre y et x, ces modèles de régression intègrent des polynômes dans leurs équations

2) THÉORÈME:
(RÉGRESSION POLYNOMIALE PAR LA MÉTHODE DES MOINDRES CARRÉS)

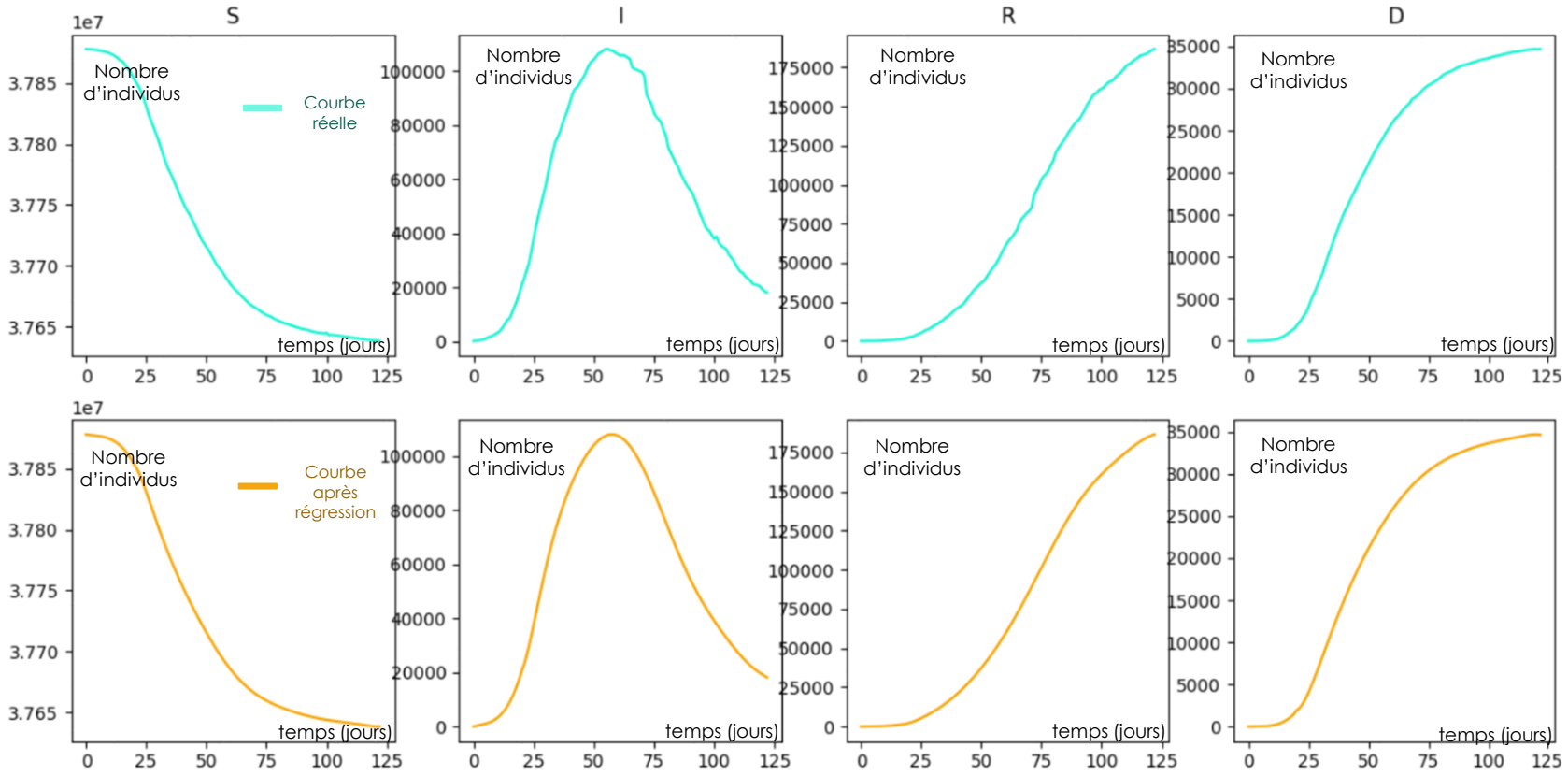
Le modèle général de régression polynomiale peut être développé par la méthode des moindres carrés. La méthode des moindres carrés vise à minimiser la variance entre les valeurs estimées par le polynôme, et le valeur attendus des données.

Ecrivons: $y = a_k x^k + a_{k-1} x^{k-1} + \ \dots \ + a_1 x + a_0 + \epsilon$
où k est le degré du polynôme de régression, ϵ est l'erreur et (a_k, $a_{k-1}, \dots, a_1, a_0$) sont les coefficients du polynômes.

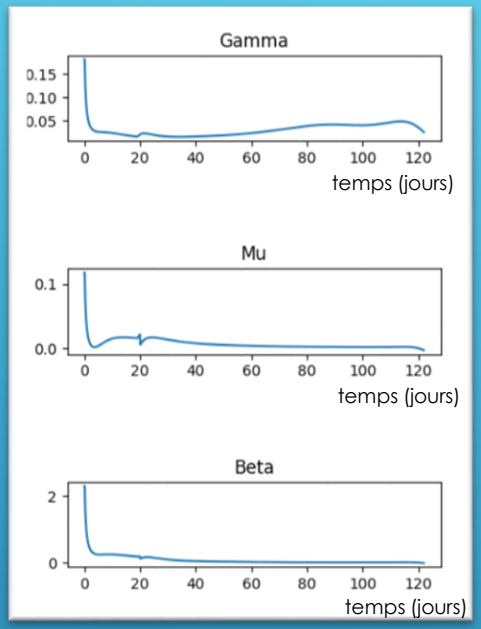
Les coefficients du polynôme de régression $(a_k, a_{k-1}, \ \dots \ , a_1, a_0)$ peuvent être déterminés en résolvant le système d'équations linéaires:

$$\begin{bmatrix} N & \sum_{i=1}^{N} x_i & \cdots & \sum_{i=1}^{N} x_i^k \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 & \cdots & \sum_{i=1}^{N} x_i^{k+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^k & \sum_{i=1}^{N} x_i^{k+1} & \cdots & \sum_{i=1}^{N} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^k y_i \end{bmatrix}$$

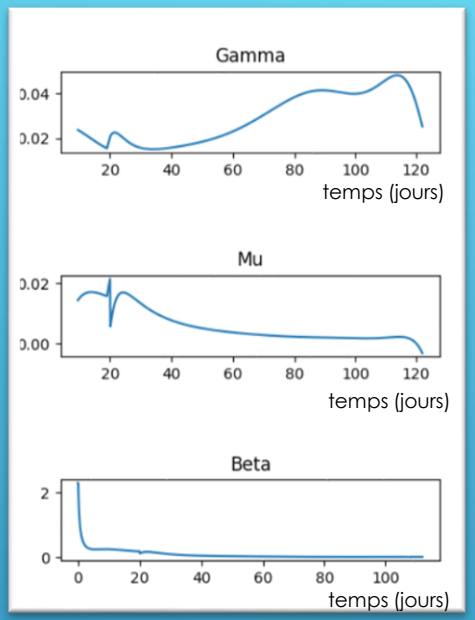
COURBES RÉELLES ET RÉGRESSION POLYNOMIALE



$$\begin{cases} \gamma = \dfrac{\frac{dR}{dt}}{I} \\ \mu = \dfrac{\frac{dD}{dt}}{I} \\ \beta = \dfrac{-N.\frac{dS}{dt}}{SI} \end{cases}$$



2) On élimine les artéfacts présents aux débuts des courbes de γ(t) et μ(t), et une partie à la fin de β(t)



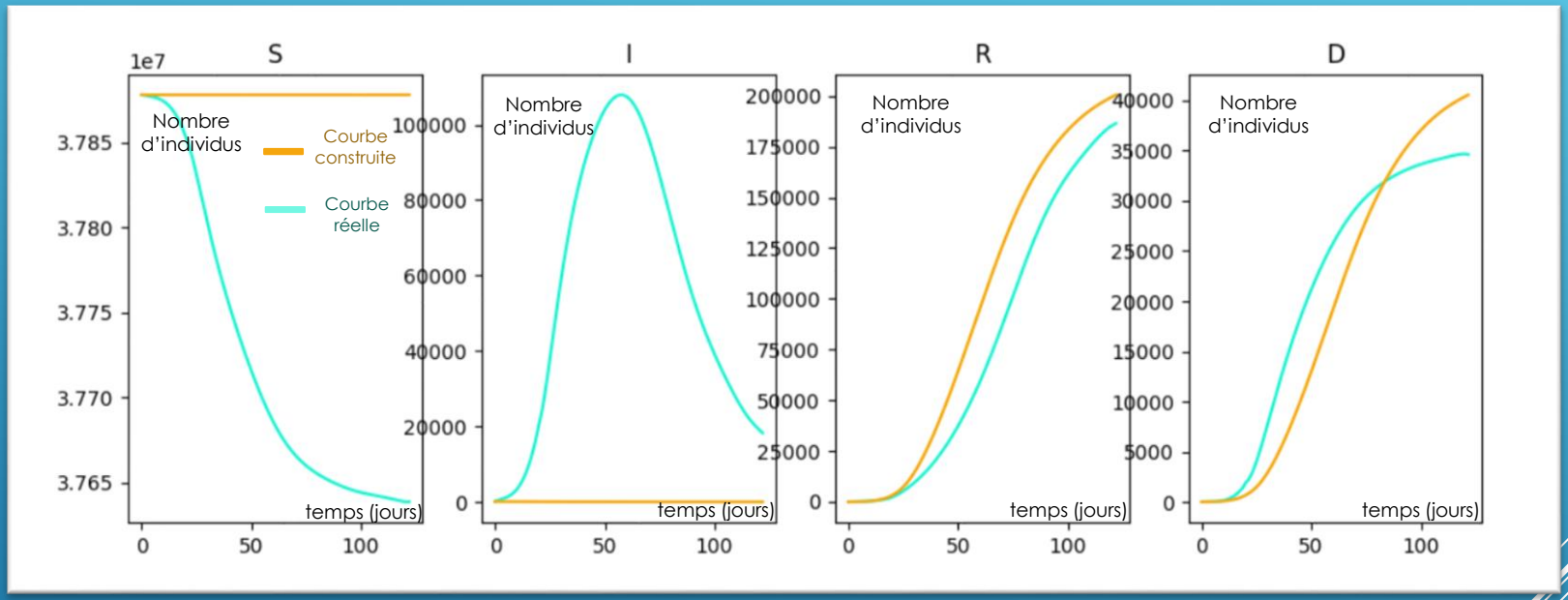
3) On récupère les moyennes sur les intervalles considérés

$$\begin{cases} \gamma = 0.00589605 \\ \mu = 0.02914553 \\ \beta = 1.038193e-05 \end{cases}$$

Détermination des constantes:

1) On trace des courbes fonctions du temps à partir des équations précédemment déterminées

TENTATIVE DE RECONSTRUIRE LES COURBES À PARTIR DES ÉQUATIONS



PROBLÈME:

Bien que les courbes obtenues pour R et D soient convenables, celles de S et I en revanche sont très erronées.

On peut expliquer cela par la supposition que β est constante au cours du temps. En effet, β caractérise à la fois la contagiosité de la maladie, mais aussi le comportement social. Il est donc prévisible que β varie énormément en période de confinement par exemple, et est intimement liée aux mesures de distanciation sociale.

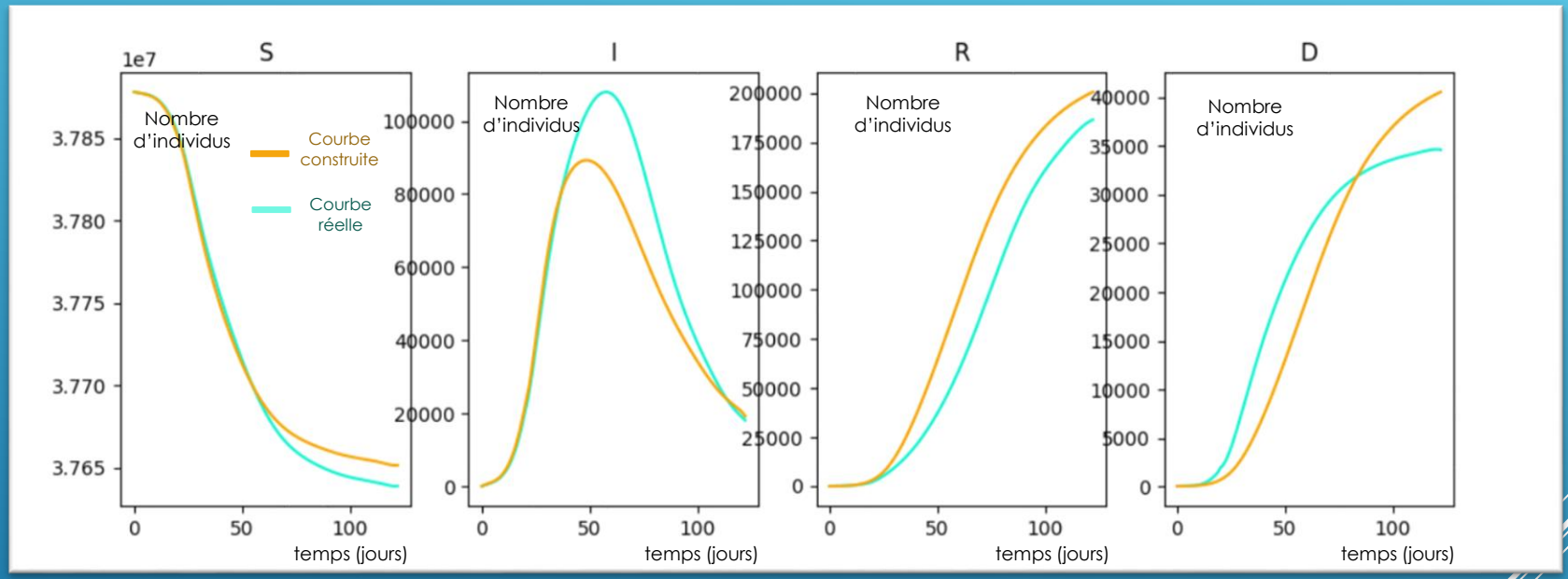
Plutôt que de considérer la moyenne de β, on maintiendra alors β variable au cours du temps afin de préserver l'information sur le comportement social.

$$\begin{cases} \dfrac{dS}{dt} = -\dfrac{\beta(t)}{N}.SI \\ \dfrac{dI}{dt} = \dfrac{\beta(t)}{N}.SI - \gamma.I - \mu.I \\ \dfrac{dR}{dt} = \gamma.I \\ \dfrac{dD}{dt} = \mu.I \end{cases}$$

L'Italie opère un confinement graduel du pars du 1er mars 2020 au 11 mars 2020
(Source: https://en.wikipedia.org/wiki/COVID-19_pandemic_in_Italy (Nationwide measures))

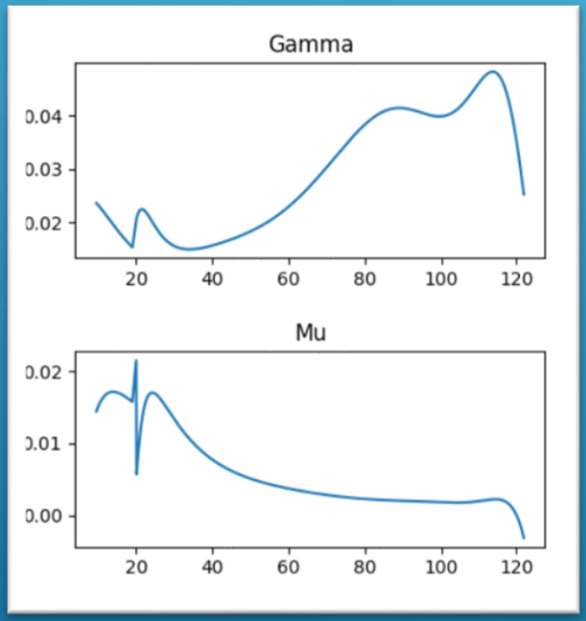
ce qui explique la chute brusque de β durant les premiers jours.

COURBES OBTENUES



Evaluation des constantes et grandeurs caractéristiques dans un cas réel: L'Italie

(Méthode 2)



$$\begin{cases} \gamma \epsilon [0{,}01\ ;\ 0{,}05] \\ \mu \epsilon [0\ ;\ 0{,}02] \end{cases}$$

Explicitation de la méthode 2

On prend:

$\begin{cases} \gamma \in [0{,}01\ ;\ 0{,}05] \\ \mu \in [0\ ;\ 0{,}02] \end{cases}$

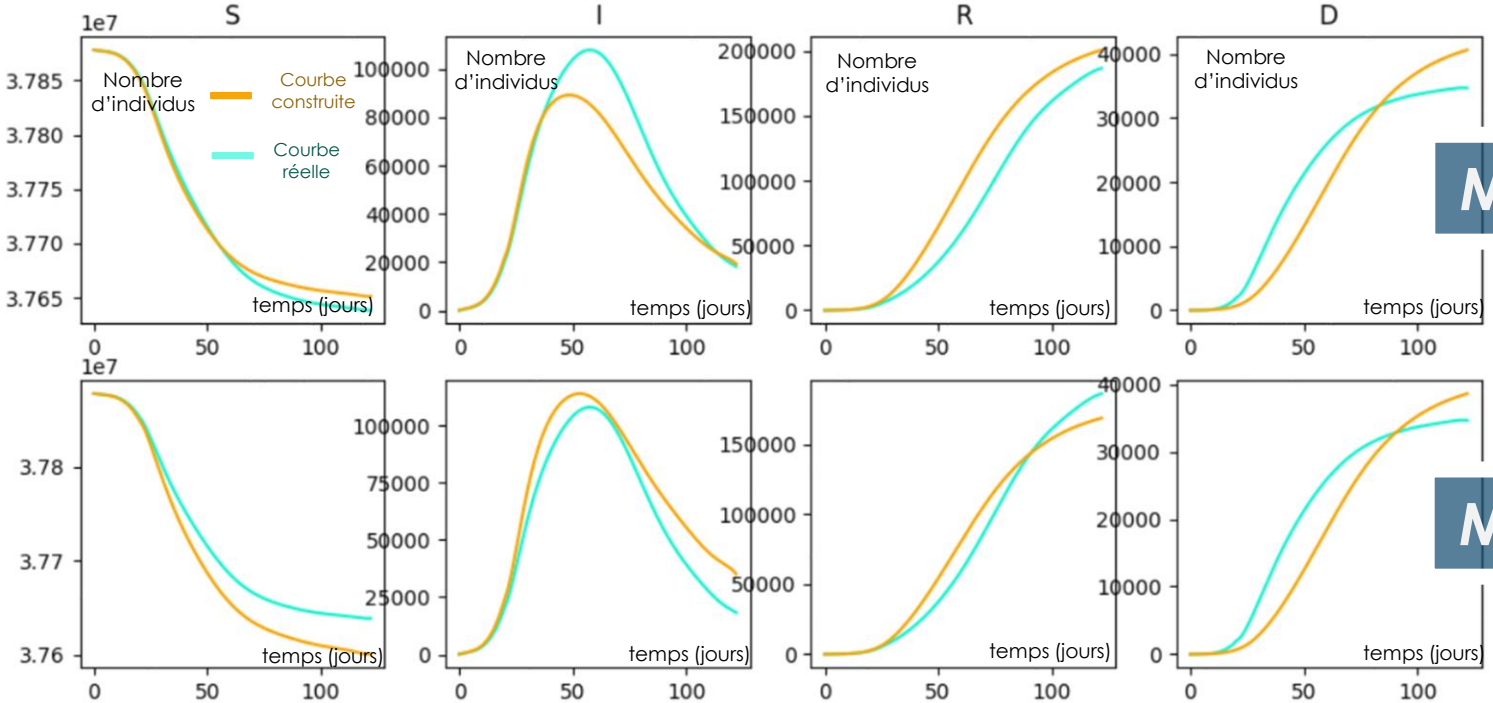
$$\begin{cases} \dfrac{dR_{construite}}{dt} = \gamma . I \\ \dfrac{dD_{construite}}{dt} = \mu . I \end{cases}$$

$$\begin{cases} \boldsymbol{R_{construite}(t) = R_{réelle}(t_0) + \gamma . \int_{t_0}^{t} I(u)du} \\ \boldsymbol{D_{construite}(t) = D_{réelle}(t_0) + \mu . \int_{t_0}^{t} I(u)du} \end{cases}$$

$$\begin{cases} \boldsymbol{d_\gamma = \int_{t_0}^{t} |R_{construite}(u) - R_{réelle}|\, du} \\ \boldsymbol{d_\mu = \int_{t_0}^{t} |D_{construite}(u) - D_{réelle}|\, du} \end{cases}$$

$\begin{cases} \boldsymbol{\gamma} = 0.02455 \\ \boldsymbol{\mu} = 0.00561 \end{cases}$

COMPARAISON DES COURBES OBTENUES PAR LES MÉTHODES 1 ET 2

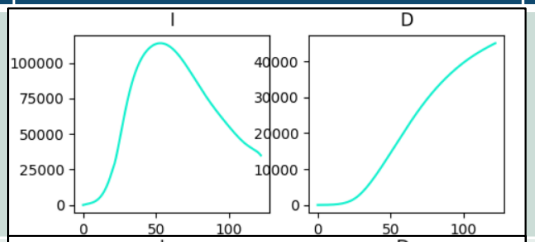
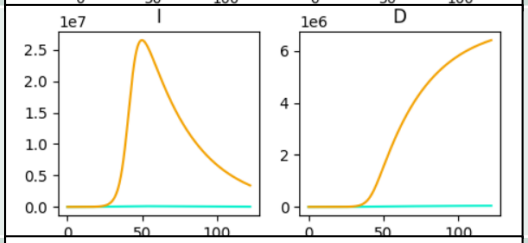
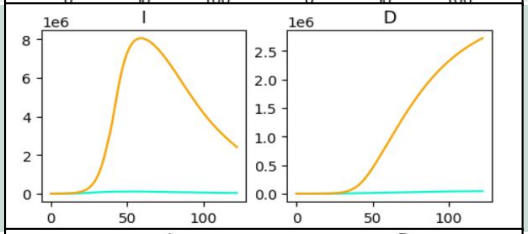
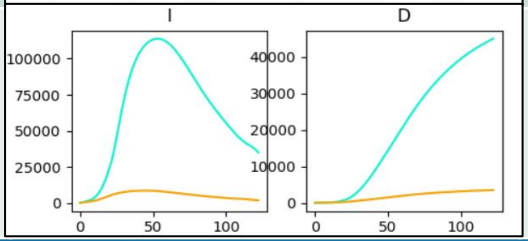


b) Modélisation des mesures de distanciation sociale dans le cadre du modèle adopté

- γ et μ représentent respectivement le taux de guérison et le taux de mortalité, les deux étant des caractéristiques de la maladie et des traitements mis en place, et n'ont pas de corrélation avec les mesures de distanciation sociale.

- β en revanche caractérise comme mentionné précédemment à la fois la contagiosité de la maladie, et le contact entre les individus.

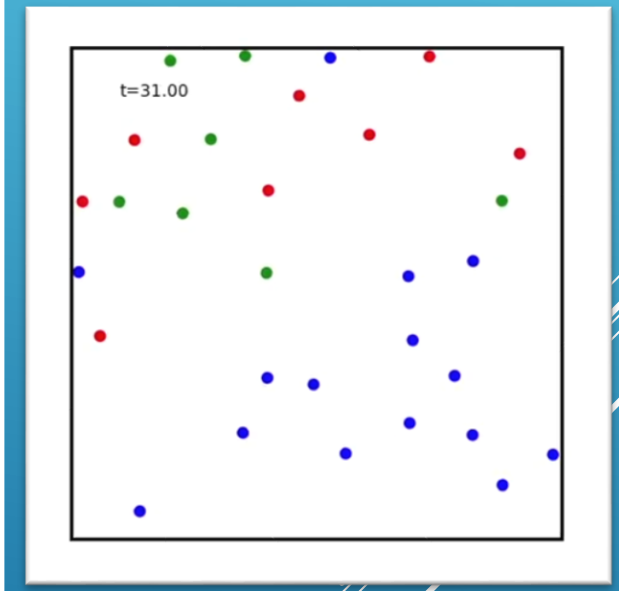
- Afin de modéliser les mesures de distanciation sociale, on peut alors agir directement sur β et observer l'effet sur la prédiction.

$\beta(t)$ considéré	Courbes de I et de D modifiées / Courbes réelles Courbe initiale — Courbe avec β modifié	Pic d'infections	Nombre de décès
$\beta_0(t)$ précédemment obtenu		113820 infectés (0,3% de la population)	45062 décès
β = à la valeur moyenne de $\beta_0(t)$ hors période de confinement		26535229 infectés (70% de la population)	6408826 décès
Période de confinement graduel rallongée (x2 durée initiale)		8042338 infectés (21% de la population)	2718663 décès
Période de confinement graduel raccourcie (x0,5 durée initiale)		8534 infectés (0,02% de la population)	3494 décès

3) MODÈLE GRAPHIQUE / SIMULATION

a) Hypothèses du modèle

- On considère une population constante, se plaçant dans un espace fermé (pas de mouvement vers ou hors de l'espace considéré)
- Un individu contaminé aura une probabilité de contaminer un individu qui lui est proche (dans un rayon autour de lui), à chaque instant.
- Une fois un certain intervalle de temps écoulé, un individu contaminé se rétablit. Il ne contaminera alors plus personne, et ne contractera pas à nouveau la maladie.
- On ne distinguera pas entre les individus rétablis et les individus décédés. (Modèle SIR simple)
- Les individus se déplacent de façon aléatoire au sein de l'espace auquel ils sont restreints, sans se soucier de possibles collisions.

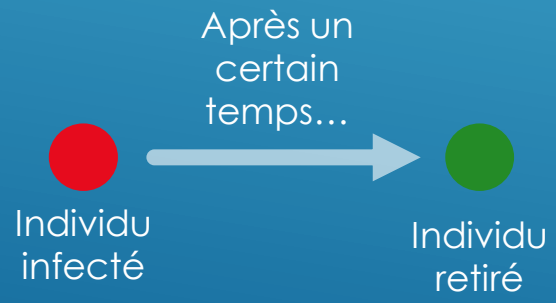


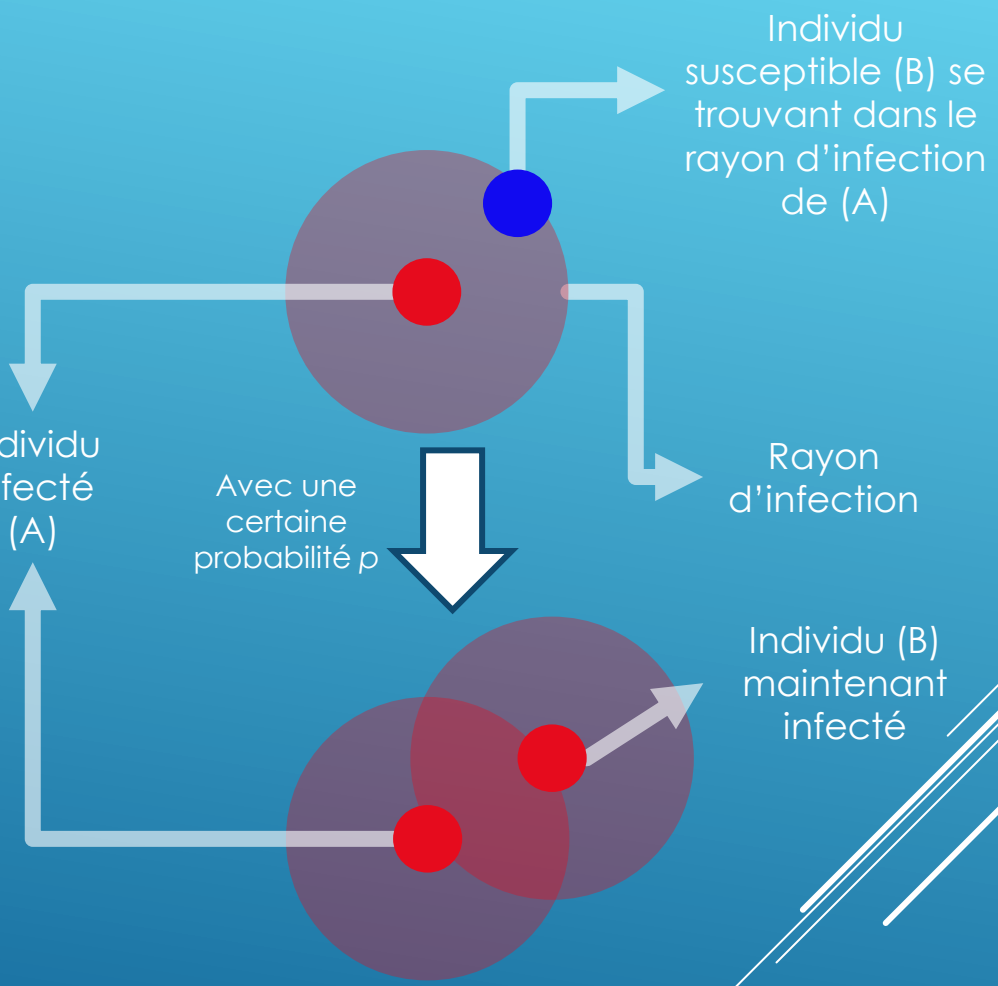
b) i) Détails du modèle

Susceptible

Infecté

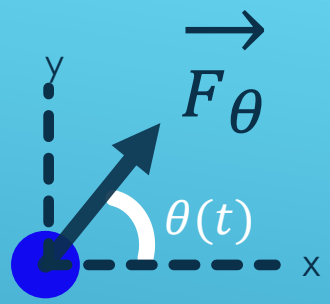
Retiré



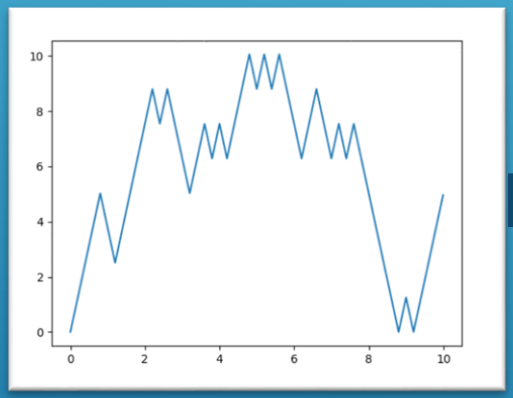


ii) Implémentation

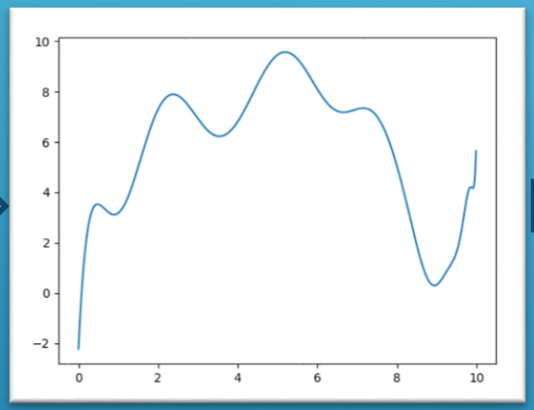
Génération du mouvement aléatoire



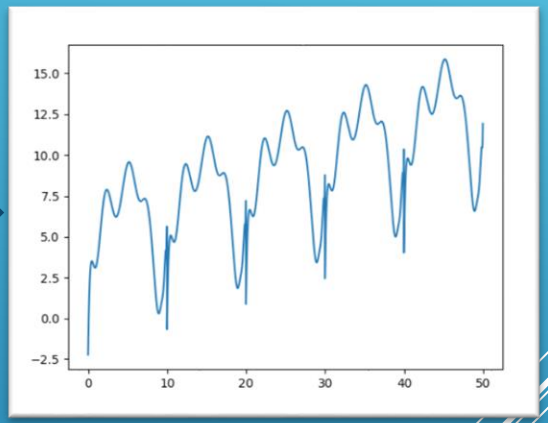
$$\overrightarrow{F_\theta} = F_0\overrightarrow{u_\theta} = \begin{pmatrix} F_0\cos(\theta(t)) \\ F_0\sin(\theta(t)) \end{pmatrix}$$



1) On génère une fonction du temps aléatoire, continue et linéaire par morceaux



2) On opère une régression polynomiale sur la fonction obtenue



3) On concatène la fonction plusieurs fois de suite en incrémentant de $\pi/2$ à chaque fois, jusqu'à couvrir l'intervalle désiré

Calcul de la position à chaque instant

D'après le Principe Fondamental de la Dynamique:

$$m\vec{a} = \overrightarrow{F_\theta} - h\vec{v}$$

$$\begin{cases} a_x(t) = \dfrac{F_0}{m}\cos\big(\theta(t)\big) - \dfrac{h}{m}v_x(t) \\ a_y(t) = \dfrac{F_0}{m}\sin\big(\theta(t)\big) - \dfrac{h}{m}v_y(t) \end{cases}$$

On calcule ensuite la vitesse et la position à l'instant t+dt en considérant que:

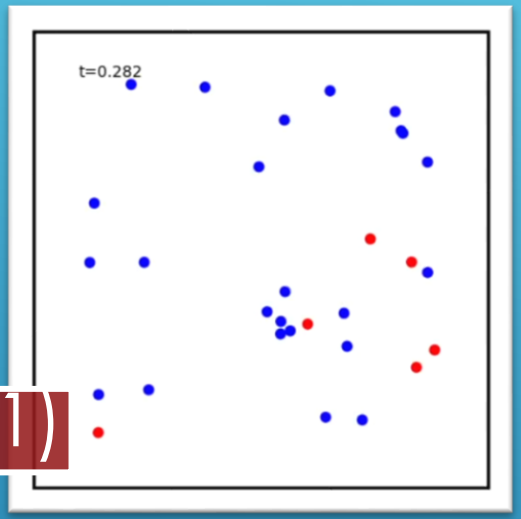
$$\begin{cases} v_x(t+dt) \approx v_x(t) + dt * a_x(t) \\ v_y(t+dt) \approx v_y(t) + dt * a_y(t) \end{cases}$$

et

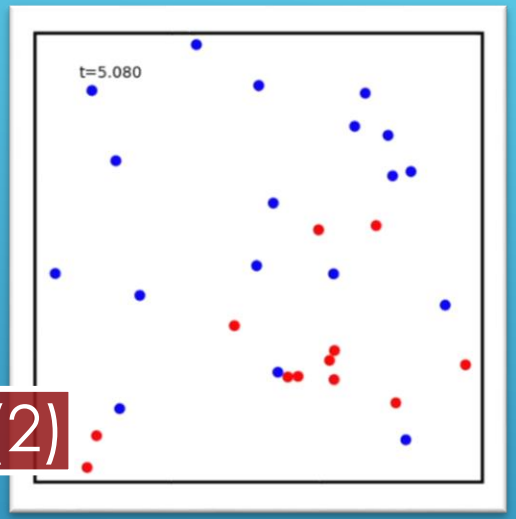
$$\begin{cases} x(t+dt) \approx x(t) + dt * v_x(t) + \dfrac{dt^2}{2} * a_x(t) \\ y(t+dt) \approx y(t) + dt * v_y(t) + \dfrac{dt^2}{2} * a_y(t) \end{cases}$$

Résultats:

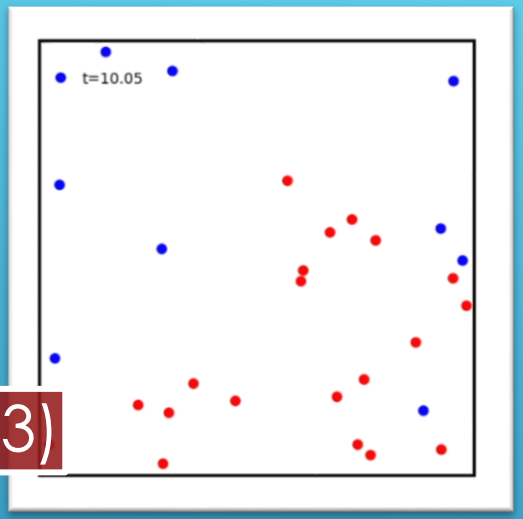
Lien vers l'animation: TIPE 42655 - Simulation SIR - YouTube



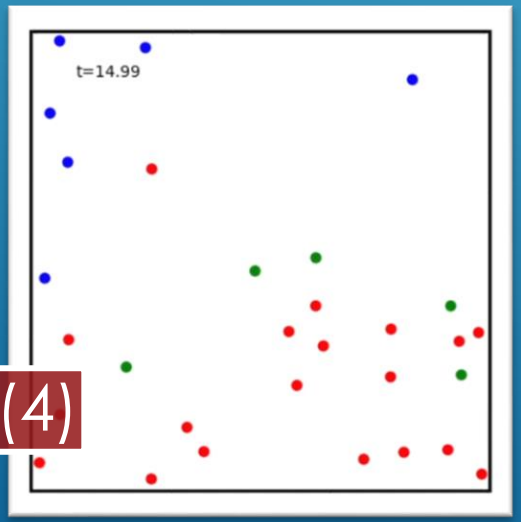
(1)



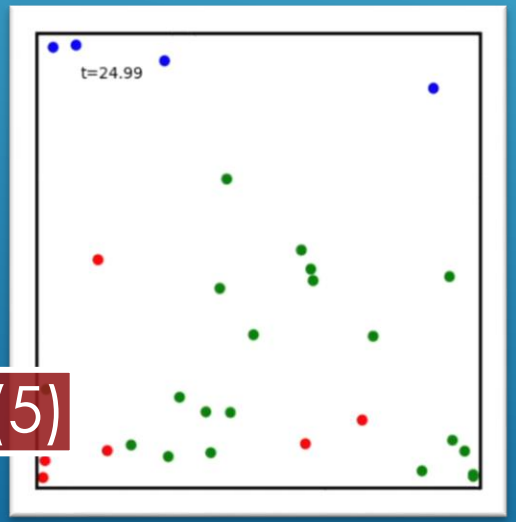
(2)



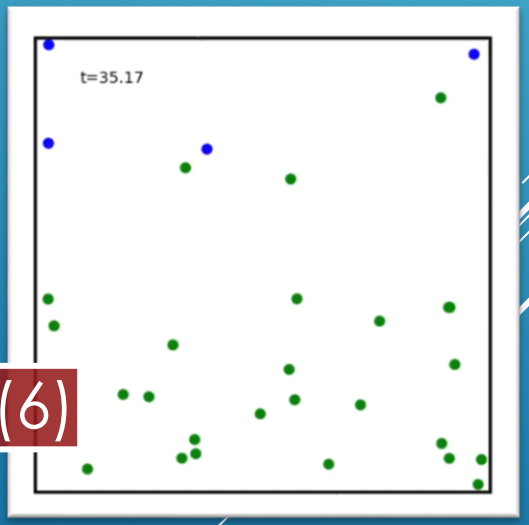
(3)



(4)

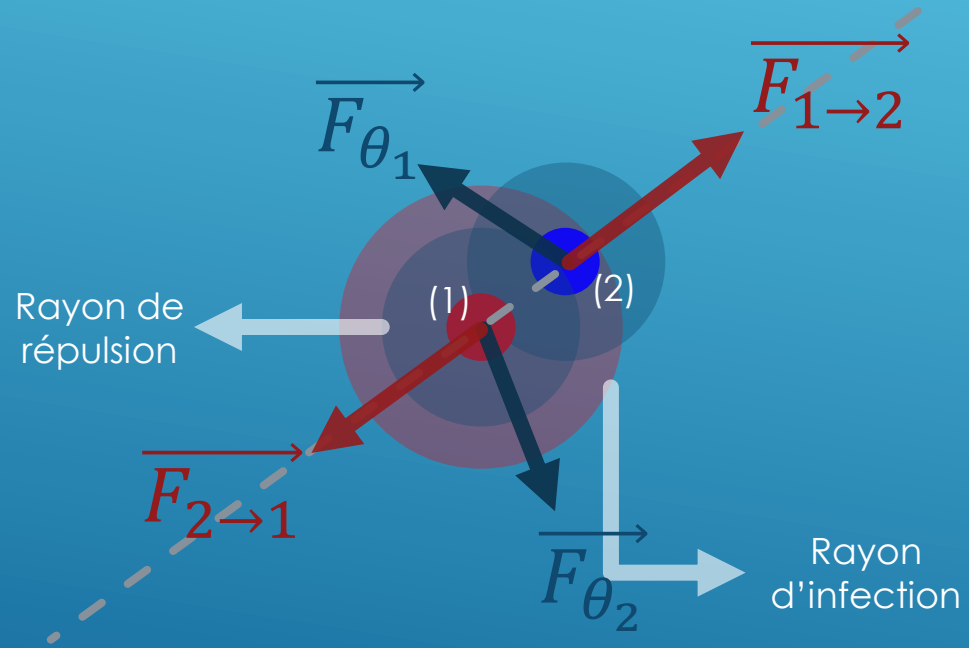


(5)



(6)

c) Modélisation et implémentation des mesures de distanciation sociale

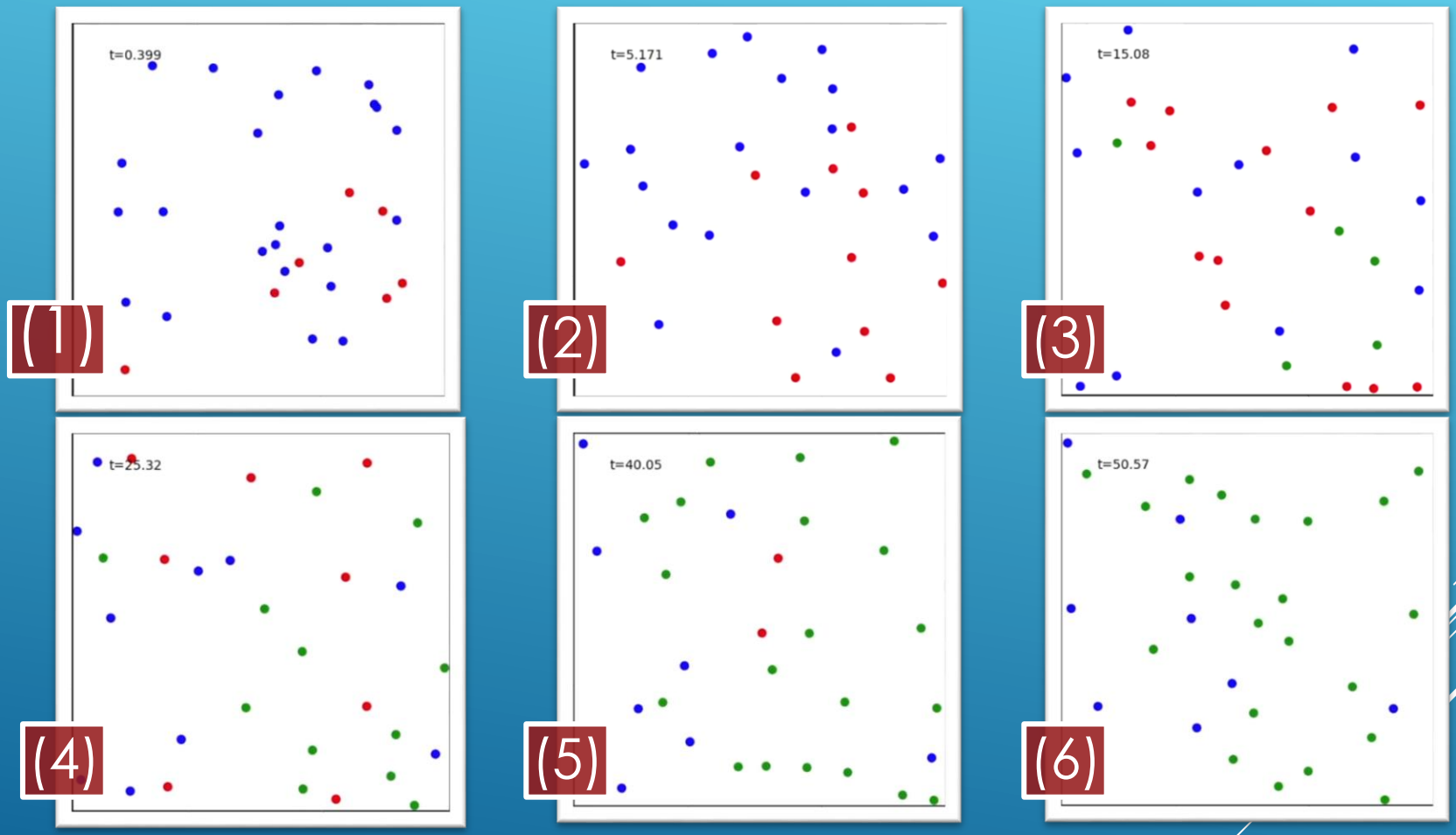


$$\overrightarrow{F_{1\to2}} = -\overrightarrow{F_{2\to1}} = \frac{\propto}{d}\overrightarrow{u_{1\to2}}$$

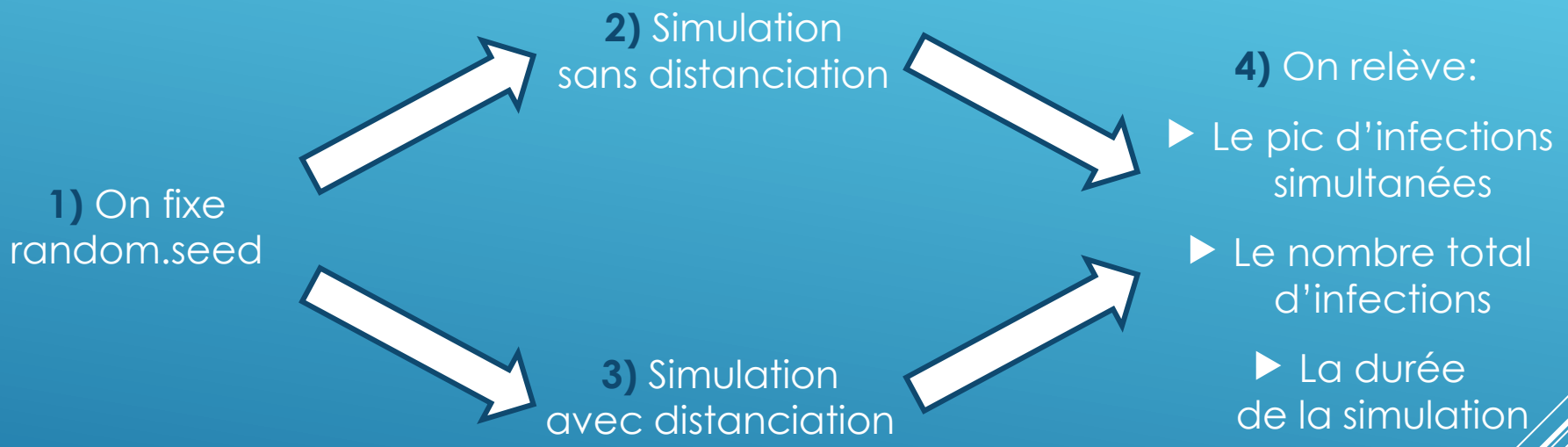
$$\begin{cases} d\text{: } distance\ entre\ (1)\ et\ (2) \\ \propto\text{: } constante \\ \overrightarrow{u_{1\to2}}\text{: } vecteur\ unitaire\ orienté\ de\ (1)\ vers\ (2) \end{cases}$$

Résultats:

Lien vers l'animation: TIPE 42655 - Simulation SIR avec répulsion - YouTube



d) Expérimentation et interprétations



Condition de fin de la simulation: Absence d'individus infectés

Taille de la zone= 10x10
Rayon d'infection= 1
Rayon de répulsion= 0,75
Temps avant guérison=15s
p=1/120
Intervalle=20ms

Seed utilisée	Pic d'infections simultanées Sans/Avec distanciation		Nombre total d'infections Sans/Avec distanciation		Durée de la simulation Sans/Avec distanciation	
1	15	10	17	12	47s	37s
2	19	11	25	11	52s	30s
3	19	11	26	11	69s	30s
4	19	9	27	9	47s	25s
5	21	9	23	9	31s	25s

4) ANNEXE (BASE DE DONNÉES)

```
#Base de données:

#   Italie:
#   Source: https://www.worldometers.info/coronavirus/country/italy/

#   Infectésx: Du 24 Février 2020 au 25 Juin 2020 (Cas actifs)

I_Italy=[220,310,455,593,822,1049,1578,1837,2265,2709,3299,3919,5064,6391,7991,8518,10593,12842,14958,17753,20607,23077,
        26066,28711,33191,37859,42672,46625,50396,53995,57469,61956,66352,69997,73806,75444,77545,80478,82951,85283,
        88166,91137,93074,93946,95133,96741,98129,100121,102102,103460,104130,105252,106436,106795,107597,108077,108047,
        107509,107494,106630,106302,105615,105867,105571,104957,104403,101286,100672,100428,99898,99696,98176,91218,89307,
        87637,84507,82983,82144,80916,78098,76074,71691,69801,67958,66155,64725,62340,60542,58899,57324,56163,54865,52501,
        50519,47529,45714,43223,41603,40893,39416,37949,38819,36492,35390,34773,34239,32376,31211,30136,28492,26977,25761,
        25394,24050,23405,22579,21417,21085,20844,20508,19441,18522,18170]


#   Rétablis: Du 24 Février 2020 au 25 Juin 2020 (Nouvelles guérisons par jour)
R_daily_Italy=[1,1,1,42,1,4,33,66,11,116,138,109,66,33,102,281,41,214,181,528,370,415,192,1087,416,691,945,954,409,896,1039,
               1001,590,1437,648,1594,1112,1121,1434,1484,1241,821,1024,1559,2104,1984,1990,2084,1681,1227,1699,964,2077,2569,
               2207,2133,1826,2730,2950,3040,2929,2628,1812,1700,2323,2317,4704,2310,1669,1744,1228,2358,8033,3038,2754,4018,
               2160,1404,2458,3510,2754,4929,2611,2372,2155,2080,2888,2284,2165,2125,1643,1506,2683,2449,3514,2245,2796,1879,
               850,1741,848,959,1891,1300,761,749,2067,1296,1402,1751,1784,1509,642,1520,931,1092,1366,547,441,534,1162,1079,614]
R_Italy=[]
S=0
for i in range(len(R_daily_Italy)):
    S+=R_daily_Italy[i]
    R_Italy.append(S)

#   Décédés: Du 24 Février 2020 au 25 Juin 2020
D_Italy=[7,10,12,17,21,29,34,52,79,107,148,197,233,366,463,631,827,1016,1266,1441,1809,2158,2503,2978,3405,4032,4825,5476,6077,
        6820,7503,8165,9134,10023,10779,11591,12428,13155,13915,14681,15362,15887,16523,17127,17669,18279,18849,19468,19899,20465,
        21067,21645,22170,22745,23227,23660,24114,24648,25085,25549,25969,26384,26644,26977,27359,27682,27967,28236,28710,28884,29079,
        29315,29684,29958,30201,30395,30560,30736,30911,31106,31368,31610,31763,31908,32007,32169,32330,32486,32616,32735,32785,32877,32955,
        33072,33142,33229,33340,33415,33475,33530,33601,33689,33774,33846,33899,33964,34043,34114,34167,34223,34301,34345,34371,34405,34448,
        34514,34561,34610,34634,34657,34675,34644,34678]
```

ALGORITHME DE RÉGRESSION POLYNOMIALE

```python
def RegressionPolynomiale(X,Y,m, retour="points"):
    n=len(X)
    S=[]
    for j in range(2*m+1):
        S.append(0)
        for i in range(n):
            S[j]+=X[i]**j
    T=[]
    for j in range(m+1):
        T.append(0)
        for i in range(n):
            T[j]+=(X[i]**j)*Y[i]
    A=np.zeros(shape=(m+1,m+1),dtype=float)
    for i in range(m+1):
        for j in range(m+1):
            A[i,j]=S[i+j]
    B=np.zeros(shape=(m+1,1),dtype=float)
    for i in range(m+1):
        B[i,0]=T[i]
    coef=np.linalg.solve(A,B)

    a=[coef[i] for i in range(m+1)]
    def f_Y(x):
        S=0
        for i in range(m+1):
            S+=a[i]*(x**i)
        return S

    Xn=np.linspace(X[0],X[-1],n*10)
    Yn=[f_Y(x) for x in Xn]

    if retour=="fonction":
        return f_Y
    else:
        return Xn,Yn
```

RÉGRESSION POLYNOMIALE DES DONNÉES RÉELLES

```
101  # Régression polynomiale de I:
102
103  T0=list(range(len(I_Italy)))
104  T1,I1=RegressionPolynomiale(T0[:20],I_Italy[:20],3)
105  T2,I2=RegressionPolynomiale(T0[20:],I_Italy[20:],10)
106  T=np.concatenate([T1,T2])
107  I=I1+I2
108
109  # Régression polynomiale de R:
110
111  T1,R1=RegressionPolynomiale(T0[:20],R_Italy[:20],3)
112  T2,R2=RegressionPolynomiale(T0[20:],R_Italy[20:],10)
113  R=R1+R2
114
115  # Régression polynomiale de D:
116
117  T1,D1=RegressionPolynomiale(T0[:20],D_Italy[:20],3)
118  T2,D2=RegressionPolynomiale(T0[20:],D_Italy[20:],10)
119  D=D1+D2
120
121  # On calcule S par la formule: S+I+R+D=N (population constante)
122  S=[]
123  for i in range(len(I)):
124      S.append(N_Italy-(I[i]+R[i]+D[i]))
125
```

TRAÇAGE DES COURBES RÉELLES ET DE RÉGRESSION

```
273   fig, axes = plt.subplots(nrows=2, ncols=4)
274
275
276   axes[0,0].plot(T0, S_Italy,color="#0ff1ce")
277   axes[0,0].set_title("S")
278   axes[0,1].plot(T0, I_Italy,color="#0ff1ce")
279   axes[0,1].set_title("I")
280   axes[0,2].plot(T0, R_Italy,color="#0ff1ce")
281   axes[0,2].set_title("R")
282   axes[0,3].plot(T0, D_Italy,color="#0ff1ce")
283   axes[0,3].set_title("D")
284   axes[1,0].plot(T, S,color="#f4a201")
285   axes[1,1].plot(T, I,color="#f4a201")
286   axes[1,2].plot(T, R,color="#f4a201")
287   axes[1,3].plot(T, D,color="#f4a201")
288
289
290
291   plt.show()
```

ÉVALUATION DES CONSTANTES (MÉTHODE 1) (COURBES FONCTION DU TEMPS ET ÉLIMINAT DES ARTÉFACTS)

```
140  # Méthode 1
141
142  # Evaluation de l'évolution des paramètres g, m et b au cours du temps:
143  gt=[]
144  mt=[]
145  bt=[]
146  for i in range(1,len(T)):
147      gt.append( ((R[i]-R[i-1])/(T[i]-T[i-1])) /I[i] )
148      mt.append( ((D[i]-D[i-1])/(T[i]-T[i-1])) /I[i] )
149      bt.append( -(((S[i]-S[i-1])/(T[i]-T[i-1]))*N_Italy/I[i]/S[i]) )
150
151  gt=[gt[0]]+gt
152  mt=[mt[0]]+mt
153  bt=[bt[0]]+bt
154
155
156  # On trouve que g et m varient de façon érratique durant les premiers jours.
157  # Afin d'éliminer ce bruit, on ne considèrera que les parties stables de m(t) et g(t)
158
159
160  T1=T[100:]
161  gt,mt=gt[100:],mt[100:]
162  T2=T[:-100]
163  bt=bt[:-100]
164
165
166  fig, axes = plt.subplots(nrows=3, ncols=1)
167  axes[0].plot(T1,gt)
168  axes[0].set_title("Gamma")
169  axes[1].plot(T1,mt)
170  axes[1].set_title("Mu")
171  axes[2].plot(T2,bt)
172  axes[2].set_title("Beta")
173
174  fig.tight_layout(pad=3.)
175
176  plt.show()
```

```
178    mmoy=0
179    gmoy=0
180    bmoy=0
181    for i in range(1,len(gt)):
182        mmoy+=(mt[i]+mt[i-1])*(T1[i]-T1[i-1])/2
183        gmoy+=(gt[i]+gt[i-1])*(T1[i]-T1[i-1])/2
184    for i in range(1,len(bt)):
185        bmoy=+(bt[i]+bt[i-1])*(T2[i]-T2[i-1])/2
186
187    mmoy/=T1[-1]-T1[0]
188    gmoy/=T1[-1]-T1[0]
189    bmoy/=T2[-1]-T2[0]
190
191    print(mmoy)
192    print(gmoy)
193    print(bmoy)
```

ÉVALUATION DES MOYENN

```python
# Méthode 2:

def genR(g): # Génère la courbe de R en fonction de R(t=0) et g
    R0=[R[0]]
    int_I=0
    for i in range(1,len(R)):
        int_I+=(I[i]+I[i-1])*(T[i]-T[i-1])/2
        R0.append(R0[0]+g*int_I)
    return R0

def genD(m): # Génère la courbe de D en fonction de D(t=0) et m
    D0=[D[0]]
    int_I=0
    for i in range(1,len(R)):
        int_I+=(I[i]+I[i-1])*(T[i]-T[i-1])/2
        D0.append(D0[0]+m*int_I)
    return D0

def distance(A,B,T): # Calcule la distance entre 2 courbes
    S=0
    for i in range(1,len(T)):
        S+=(abs(A[i]-B[i])+abs(A[i-1]-B[i-1]))*(T[i]-T[i-1])/2
    return S

```

ÉVALUATION DES CONSTANTES (MÉTHODE 2) (PAGE 1)

```
209  m=0.
210  mmin=0.
211  D0=genD(m)
212  emin=distance(D0,D,T)
213
214  while (m<0.02):
215      D0=genR(m)
216      e=distance(D0,D,T)
217      if (e<emin):
218          emin=e
219          mmin=m
220      m+=1e-5
221
222  m=mmin
223
224
225  g=0.01
226  gmin=0.01
227  R0=genR(g)
228  emin=distance(R0,R,T)
229
230  while (g<0.05):
231      R0=genR(g)
232      e=distance(R0,R,T)
233      if (e<emin):
234          emin=e
235          gmin=g
236      g+=1e-5
237
238  g=gmin
239
```

ÉVALUATION DES CONSTANTES (MÉTHODE 2) (PAGE 2)

```
S0=[S[0]]
I0=[I[0]]
R0=[R[0]]
D0=[D[0]]

for i in range(1,len(T)):
    S0.append(S0[i-1]-(bt[i-1]*I0[i-1]*S0[i-1]/N_Italy)*(T[i]-T[i-1]))
    I0.append(I0[i-1]+((bt[i-1]*I0[i-1]*S0[i-1]/N_Italy)-gmoy*I0[i-1]-mmoy*I0[i-1])*(T[i]-T[i-1]))
    R0.append(R0[i-1]+(gmoy*I[i-1])*(T[i]-T[i-1]))
    D0.append(D0[i-1]+(mmoy*I[i-1])*(T[i]-T[i-1]))


fig, axes = plt.subplots(nrows=2, ncols=4,figsize=(3,4))
axes[0,1].plot(T, I,color="#0ff1ce")
axes[0,1].set_title("I")
axes[0,2].plot(T, R,color="#0ff1ce")
axes[0,2].set_title("R")
axes[0,3].plot(T, D,color="#0ff1ce")
axes[0,3].set_title("D")
axes[0,1].plot(T, I0,color="#f4a201")
axes[0,2].plot(T, R0,color="#f4a201")
axes[0,3].plot(T, D0,color="#f4a201")
axes[0,0].plot(T, S,color="#0ff1ce")
axes[0,0].set_title("S")
axes[0,0].plot(T, S0,color="#f4a201")
```

TRAÇAGE DES COURBES DE COMPARAISON DES 2 MÉTHODES (PAGE 1)

```
284  S0=[S[0]]
285  I0=[I[0]]
286  R0=[R[0]]
287  D0=[D[0]]
288
289  for i in range(1,len(T)):
290      S0.append(S0[i-1]-(bt[i-1]*I0[i-1]*S0[i-1]/N_Italy)*(T[i]-T[i-1]))
291      I0.append(I0[i-1]+((bt[i-1]*I0[i-1]*S0[i-1]/N_Italy)-g*I0[i-1]-m*I0[i-1])*(T[i]-T[i-1]))
292      R0.append(R0[i-1]+(g*I[i-1])*(T[i]-T[i-1]))
293      D0.append(D0[i-1]+(m*I[i-1])*(T[i]-T[i-1]))
294
295  axes[1,1].plot(T, I,color="#0ff1ce")
296  axes[1,2].plot(T, R,color="#0ff1ce")
297  axes[1,3].plot(T, D,color="#0ff1ce")
298  axes[1,1].plot(T, I0,color="#f4a201")
299  axes[1,2].plot(T, R0,color="#f4a201")
300  axes[1,3].plot(T, D0,color="#f4a201")
301  axes[1,0].plot(T, S,color="#0ff1ce")
302  axes[1,0].plot(T, S0,color="#f4a201")
303
304
305
306  plt.show()
```

TRAÇAGE DES COURBES DE COMPARAISON DES 2 MÉTHODES (PAGE 2)

```
273   #Rallongement confinement graduel x2
274   bt2=[]
275   i=1
276   while (T[i]<17):
277       bt2+=[bt[i]]*2
278       i+=1
279   bt2+=bt[i:]
280   bt2=bt2[:len(T)]
281   bt=bt2
```

ALGORITHME DE RALLONGEMENT DE β

```
275  #Raccourcissement confinement graduel x0.5
276  bt2=[]
277  i=0
278  while (T[i]<17):
279      bt2.append(bt[i])
280      i+=2
281  bt2+=bt[i:]
282  bt2=bt2[:len(T)]
283  bt=bt2
```

ALGORITHME DE RACCOURCISSEMENT DE β

```python
def multiplier(T,f,m):
    n=len(f)
    for i in range(1,m):
        f0=[x+pi/2 for x in f[-n:]]
        f=f+f0
    T=np.linspace(T[0],T[-1]*m,n*m)
    return T,f

def GenererFonctionAleatoire(duree_totale):
    T=[0]
    f=[2*pi*random.random()]
    dt=1/30
    intervalle=20
    k=intervalle
    e=1e-2
    duree=10
    for i in range(1,int(duree/dt)):
        if i>k:
            e=e*random.choice([-1,1])
            k+=intervalle
        f.append(f[i-1]+e)
        T.append(T[i-1]+dt)
    f=[5*pi*y for y in f]
    T,f=RegressionPolynomiale(T,f,20)
    m=int(duree_totale/duree)
    T,f=multiplier(T,f,m)
    return T,f

```

GÉNÉRATION DE LA FONCTION ALÉATOIRE $\theta(t)$

CLASSE POPULATION __INIT__

```
82  class Population:
83      def __init__(self,mvt_init,etat_init,T,f,F0,h,m,p,r_inf=1,r_rep=0.75,a=0.3,t_guerison=15):
84          self.mvt_init=np.asarray(mvt_init, dtype=float)
85          self.mvt=self.mvt_init.copy()
86          self.etat=copy(etat_init)
87          self.T,self.f,self.p,self.F0,self.h,self.m,self.r_inf,self.r_rep,self.a,self.t_gerison=T,f,p,F0,h,m,r_inf,r_rep,a,t_guerison
88          self.t=0
89          self.n=len(etat_init)
90          self.colors=[]
91          for i in range(len(etat_init)):
92              if (etat_init[i]==1):
93                  self.colors.append("red")
94              elif (etat_init[i]==0):
95                  self.colors.append("blue")
96              else:
97                  self.colors.append("green")
98          self.t_guerison=t_guerison
99          self.t_infection=[time()]*self.n
100
```

CLASSE POPULATION FONCTION PAS (MISE À JOUR DU TEMPS, POSITIONS, VITESSES ET INFECTIONS)

```python
    def pas(self,dt):
        # Mets à jour le temps, les positions et les vitesses
        self.t+=dt
        self.mvt[:,2:4]+=dt*self.mvt[:,4:]
        self.mvt[:,:2]+=dt*self.mvt[:,2:4]+(dt**2/2)*self.mvt[:,4:]

        # Mets à jour les guérisons:
        for i in range(self.n):
            if (self.etat[i]==1) and (time()-self.t_infection[i]>self.t_guerison):
                    self.etat[i]=2



        # Détermine les couples de points proches du rayon d'infection
        D1 = squareform(pdist(self.mvt[:, :2]))
        ind1, ind2 = np.where(D1 < self.r_inf)
        unique = (ind1 < ind2)
        ind1 = ind1[unique]
        ind2 = ind2[unique]

        # Détermine si une infection a lieu
        for i1, i2 in zip(ind1,ind2):
            if (self.etat[i1]==1 and self.etat[i2]==0) or (self.etat[i1]==0 and self.etat[i2]==1):
                roll=random.random()
                if roll<self.p:
                    if (self.etat[i1]==1):
                        self.etat[i2]=1
                        self.t_infection[i2]=time()
                    else:
                        self.etat[i1]=1
                        self.t_infection[i1]=time()
```

CLASSE POPULATION FONCTION PAS (MISE À JOUR DES COULEURS, ET PREND EN CHARGE LES COLLISIONS AVEC LES BORDS)

```
135         # Met à jour les couleurs
136         for i in range(len(self.etat)):
137             if (self.etat[i]==1):
138                 self.colors[i]="red"
139             elif (self.etat[i]==0):
140                 self.colors[i]="blue"
141             elif (self.etat[i]==2):
142                 self.colors[i]="green"
143 
144         # Détermine si un point dépasse les bords
145         depasse_x1=(self.mvt[:,0]-0.15< 0)
146         depasse_x2=(self.mvt[:,0]+0.15>10)
147         depasse_y1=(self.mvt[:,1]-0.15<0)
148         depasse_y2=(self.mvt[:,1]+0.15>10)
149 
150         self.mvt[depasse_x1,0]=0.15
151         self.mvt[depasse_x2,0]=10-0.15
152         self.mvt[depasse_y1,1]=0.15
153         self.mvt[depasse_y2,1]=10-0.15
154 
155         self.mvt[depasse_x1 | depasse_x2,2] *= -0.9
156         self.mvt[depasse_y1 | depasse_y2,3] *= -0.9
```

```
158         # Applique la force aléatoire
159         j=determinerIndex(self.t,self.T)
160         for i in range(self.n):
161             F=np.array([cos(self.f[i][j])*self.F0,sin(self.f[i][j])*self.F0])
162             self.mvt[i,4:]=F/self.m-self.h/self.m*self.mvt[i,2:4]
163 
164         # Applique la force de répulsion
165         ind1, ind2 = np.where(D1 < self.r_rep)
166         unique = (ind1 < ind2)
167         ind1 = ind1[unique]
168         ind2 = ind2[unique]
169         for i1,i2 in zip(ind1,ind2):
170             d=sqrt( (self.mvt[i1,0]-self.mvt[i2,0])**2 + (self.mvt[i1,0]-self.mvt[i2,0])**2 )
171             u=(self.mvt[i2,:2]-self.mvt[i1,:2])/d
172             self.mvt[i1,4:]+= -min(self.a/d/self.m,20)*u
173             self.mvt[i2,4:]+= min(self.a/d/self.m,20)*u
```

CLASSE POPULATION FONCTION PAS (APPLIQUE LES FORCES ALÉATOIRE ET DE RÉPULSION)

ÉTAT INITIAL

```
175  # Etat initial
176  random.seed(5)
177  n=30 #Nombre de points
178  duree=30 #Durée de l'animation en secondes
179  dt=5e-3
180  mvt_init=[[random.randint(1,8)+random.random(), random.randint(1,8)+random.random(),0,0,0,0] for _ in range(n)]
181  etat_init=[1]*5+[0]*25
182  f=[]
183  for _ in range(n):
184      T,f0=GenererFonctionAleatoire(duree)
185      f.append(f0)
186
187  population=Population(mvt_init,etat_init,T,f,4,1,1,1/120,a=10,r_rep=1)
188
```

```python
# Mise en place de l'animation
fig=plt.figure()
fig.subplots_adjust(left=0, right=1, bottom=0, top=1)
ax = fig.add_subplot(111, aspect='equal', autoscale_on=False,
                     xlim=(-1, 11), ylim=(-1, 11))
ax.axis('off')
scat=ax.scatter([],[])
bords = [0, 10, 0, 10]
rect = plt.Rectangle(bords[::2],
                     bords[1] - bords[0],
                     bords[3] - bords[2],
                     ec='none', lw=2, fc='none')
ax.add_patch(rect)


time_text=ax.text(1,9,'')
def init():
    rect.set_edgecolor('none')
    scat.set_offsets([])
    time_text.set_text('')
    return scat, rect,time_text

def animate(i):
    global population,rect,dt,ax,fig,t0
    population.pas(dt)
    time_text.set_text('t='+str(time()-t0)[:5])
    rect.set_edgecolor('k')
    scat.set_offsets(population.mvt[:,:2])
    scat.set_color(population.colors)
    return scat,rect,time_text

t0=time()
anim=animation.FuncAnimation(fig,animate,interval=20,blit=True,init_func=init,frames=5000)

plt.show()
```

ANIMATION