L'épidémiologie mathématique

Marouane IBN BRAHIM
Numéro de dossier : 1313

TIPE 2019
Thème : Les transports

Un peu d'histoire

- L'épidémiologie daterait du Ve siècle avant J.C
- Hippocrate (460 av. J.-C,377 av. J.-C) serait le premier épidémiologue
- Les mathématiques n'avaient pas leur place en épidémiologie
- Ce n'est qu'au $XVIII^e$ siècle avec les travaux de Daniel Bernoulli (1700-1782) que l'on commence à comprendre leur importance

L'épidémiologie mathématique, c'est quoi exactement ?

Consiste à modéliser mathématiquement une épidémie à travers :

- Un paramétrage
- Une mise en équations
- La résolution des équations
- La comparaison avec les statistiques réelles pour juger l'efficacité et la fidélité des modèles proposés

Définition

Les modèles compartimentaux sont une modélisation mathématique des épidémies consistant à :

- Diviser la population en plusieurs compartiments (Groupes) tels que : Susceptibles(S), Infectieux(I)...
- Décrire les interactions et les échanges entre eux

Quelques notations

- N désigne la population totale à un instant t
- S désigne le nombre de susceptibles à un instant t
- I désigne le nombre d'infectieux à un instant t
- E désigne le nombre d'éxposés à un instant t
- R désigne le nombre de personnes guéries et immunisées à un instant t
- X' désigne la dérivée d'une grandeur X par rapport au temps
- β désigne le taux de contagion
- γ désigne le taux de guérison
- μ désigne le taux de létalité

Le modèle SIS
Présentation du modèle

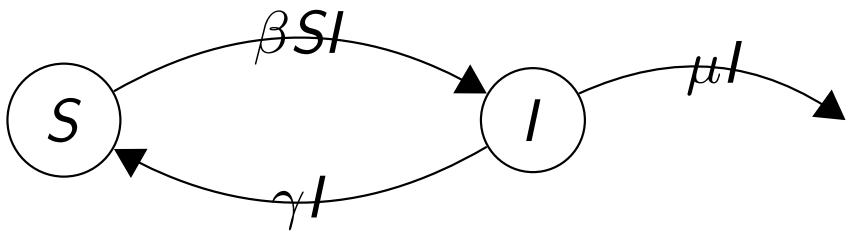
Le modèle SIS (pour Susceptible-Infectieux-Susceptible) décrit une épidémie dont les guéris ne sont pas immunisés.
Il consiste à diviser la population en deux compartiments :

- Susceptibles
- Infectieux

Le système d'équations le décrivant : $\begin{cases} S' = -\beta SI + \gamma I \quad (1) \\ I' = \beta SI - \gamma I - \mu I \quad (2) \end{cases}$

Le modèle SIS

Un schéma pour mieux visualiser



Justification :

- βN : le nombre de contact suffisant pour transmettre la maladie par personne par unité de temps
- $\beta N * \frac{S}{N} = \beta S$: Nombre de personnes infectées par infectieux par unité de temps
- βSI : Nombre de personnes infectées par unité de temps
- γI : Nombre de personnes guéries par unité de temps
- μI : Nombre de personnes mortes par la maladie par unité de temps

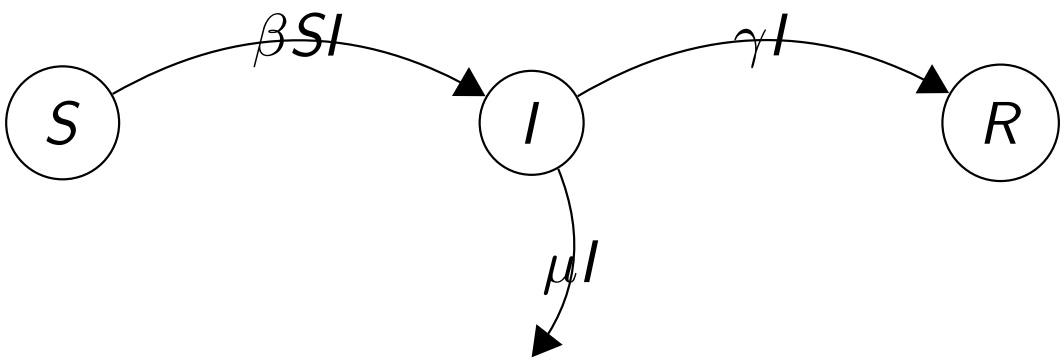
Le modèle SIR
Présentation du modèle

Le modèle SIR (pour Susceptible-Infectieux-Retiré) décrit une épidémie dont les guéris sont immunisés à vie (ou au moins pendant la période d'étude).
Il consiste à diviser la population en trois compartiments :

- Susceptibles
- Infectieux
- Retirés

Le modèle SIR

Un schéma pour mieux visualiser

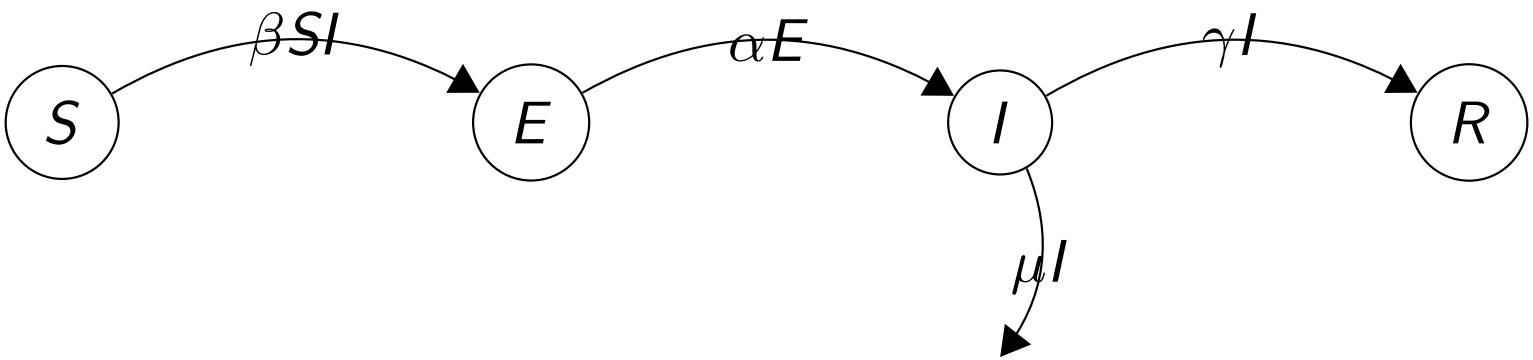


$$\begin{cases} S' = -\beta SI & (1) \\ I' = \beta SI - \gamma I - \mu I & (2) \\ R' = \gamma I & (3) \end{cases}$$

Le modèle SEIR

Présentation et schéma explicatif

Le modèle SEIR (pour Susceptible-Exposé-Infectieux-Retiré) est similaire au modèle SIR mais en prenant en compte la période d'incubation de la maladie.



$$\begin{cases} S' = -\beta SI \quad (1) \\ E' = \beta SI - \alpha E \quad (2) \\ I' = \alpha E - \mu I - \gamma I \quad (3) \\ R' = \gamma I \quad (4) \end{cases}$$

Le virus Ebola au Sierra Leone en nombres

- Une épidémie qui a duré entre novembre 2014 et mai 2016.
- Plus de 14.000 cas suspectés
- Plus de 3900 morts suspectées

Comment déterminer les constantes α, β, γ, et μ adéquates à notre situation ?

L'interpolation Polynômiale

Etant donné deux familles $(x_i)_{1\leq i\leq n+1}$ ordonnée par ordre croissant et $(y_i)_{1\leq i\leq n+1}$, il s'agit de chercher un polynôme de degré inférieur strictement à n, qui prend la valeur y_i au point x_i.

Les polynômes de Lagrange :

On pose pour $i \in [\![1\ ;\ n+1]\!]$

$$L_i(X) = \prod_{j=1, j\neq i}^{n+1} \frac{X - x_j}{x_i - x_j}$$

et :

$$P(X) = \sum_{i=1}^{n+1} y_i L_i(X)$$

Alors P interpole $(y_i)_{1\leq i\leq n+1}$ aux points $(x_i)_{1\leq i\leq n+1}$

Pour le modèle SIR

- On supposera dans toute la suite que $\mu = 0$
- On interpole la famille : "Nombre de susceptibles à l'instant t" en les points du temps
- On interpole la famille : "Nombre d'infectieux à l'instant t" en les points du temps
- $(1) \Rightarrow \beta = -\frac{S'}{SI}$
- on calcule une moyenne de β sur l'intervalle d'étude

  $i.e : \quad \beta = \frac{1}{T}\int_0^T (-\frac{S'}{SI})(t)\,\mathrm{d}t$

- En interpolant par 25 on obtient : $\beta = 1.505 \quad . \quad 10^{-8}$
- En interpolant par 30 on obtient : $\beta = 5.556 \quad . \quad 10^{-8}$

Conclusion : La méthode trop instable est à rejeter.

Approximation : Méthode des moindres carrés

- Etant donné deux familles $(x_i)_{1\leq i\leq n+1}$ ordonnée par ordre croissant et $(y_i)_{1\leq i\leq n+1}$, il s'agit de chercher une fonction f qui minimize la quantité :

$$\sum_{i=1}^{n+1}(y_i - f(x_i))^2$$

- Nous optons dans notre cas pour un polynôme de degré 2 il s'agit donc de trouver $P \in \mathbb{R}_2[X]$ tel que :

$$\sum_{i=1}^{n+1}(y_i - P(x_i))^2$$

Proposition :

- $min\left\{\sum_{i=1}^{n+1}(y_i - P(x_i))^2 / P \in \mathbb{R}_2[X]\right\}$ existe.
- Si $P(X) = a + bX + cX^2$ est un tel polynôme alors :

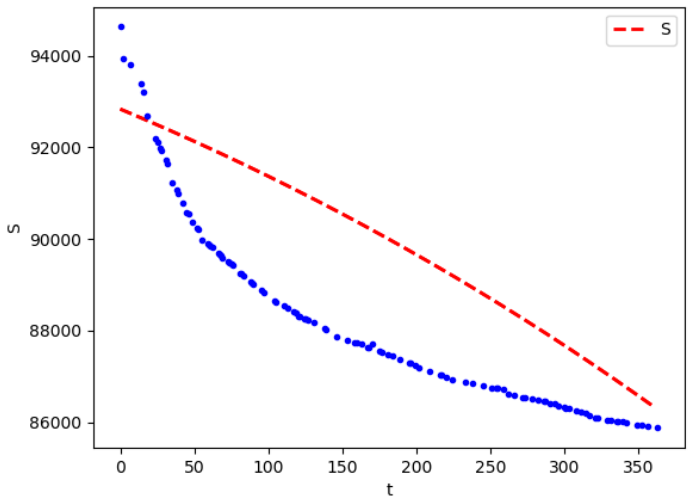
$$^tAAX = {}^tAB$$

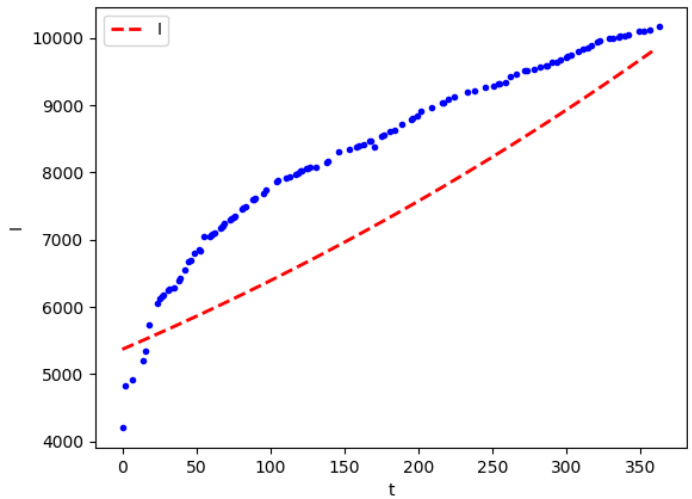
où

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 \end{pmatrix} \quad X = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad B = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n+1} \end{pmatrix}$$

Résultats pour le modèle SIR

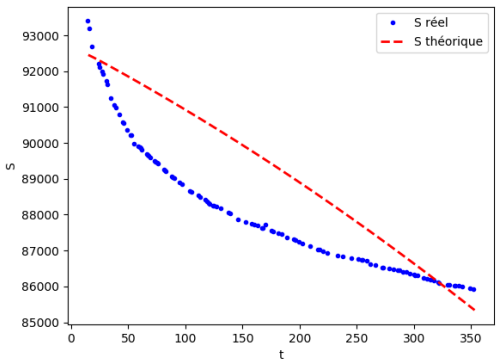
- $\beta = 2.714 \quad . \quad 10^{-8}$
- $\gamma = 7.57 \quad . \quad 10^{-4}$

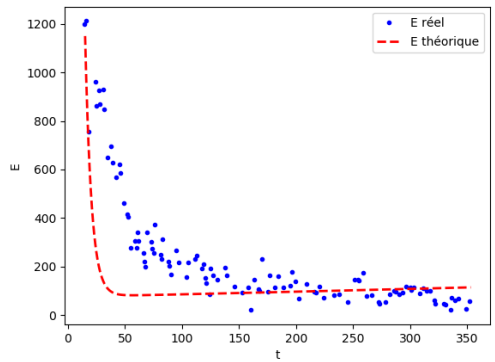


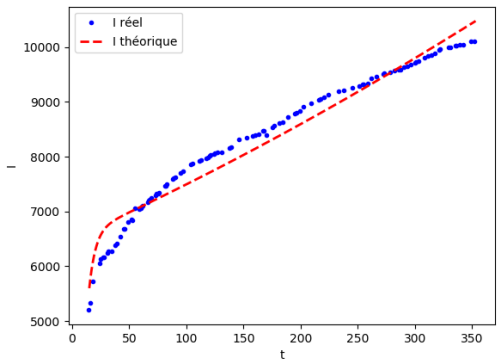


Résultats pour le modèle SEIR

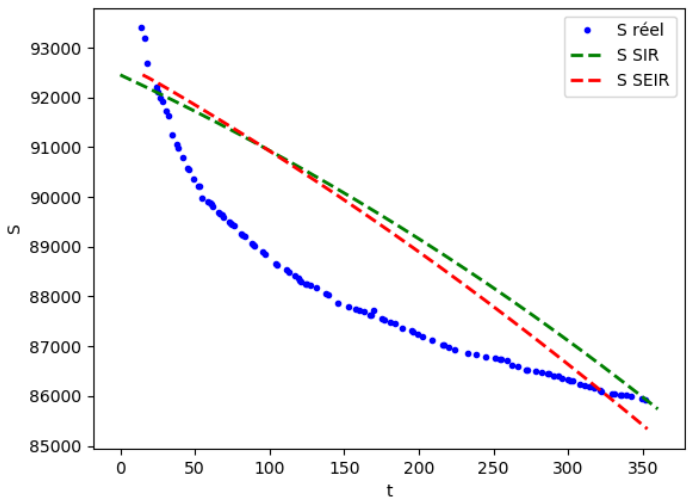
- $\beta = 2.806 \quad . \quad 10^{-8}$
- $\gamma = 1.139 \quad . \quad 10^{-3}$
- $\alpha = 0.1775$

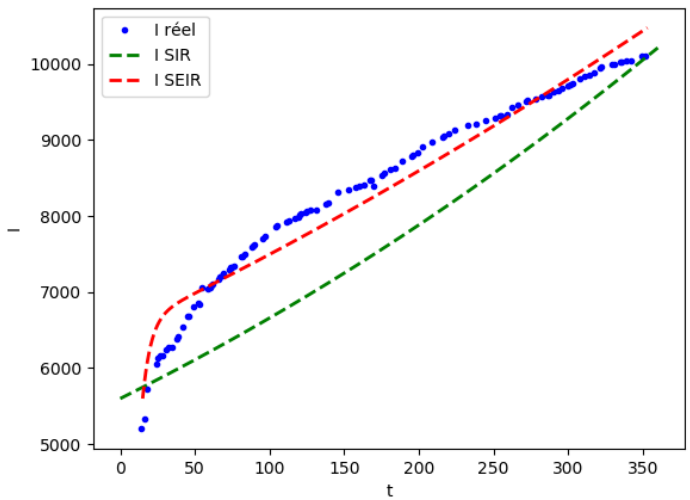






Comparaison





Merci pour votre attention !

Méthode des moindres carrés

Soit $P = a + bX + cX^2 \in \mathbb{R}_2[X]$. On a :

$$\begin{aligned}\sum_{i=1}^{n+1}(y_i - P(x_i))^2 &= \sum_{i=1}^{n+1}(y_i - a - bx_i - cx_i^2)^2 \\ &= \|AX - B\|\end{aligned}$$

où

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 \end{pmatrix} \quad X = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \quad B = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_{n+1} \end{pmatrix}$$

La matrice A est de rang 3. En effet si $X = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix}$ est solution de

$AX = 0$ alors $x_1, x_2, ..., x_{n+1}$ sont racines du polynôme :
$Q(X) = \alpha_0 + \alpha_1 X + \alpha_2 X^2$ Dans le cadre de notre étude $n + 1 > 2$ donc c'est le polynôme nul, d'où le résultat.
Soit C le projeté orthogonal de B sur $Im(A)$. A étant de rang 3. L'application $X \mapsto AX$ de $\mathcal{M}_{3,1}(\mathbb{R})$ vers $Im(A)$ est injective donc il existe un unique X_0 pour lequel le minimum est atteint. On a :

$$\begin{aligned} AX_0 = p_{Im(A)}(B) &\iff \forall X \in \mathcal{M}_{3,1}(\mathbb{R}) : AX_0 - B \perp AX \\ &\iff \forall X \in \mathcal{M}_{3,1}(\mathbb{R}) : {}^t(AX)(AX_0 - B) = 0 \\ &\iff \forall X \in \mathcal{M}_{3,1}(\mathbb{R}) : {}^tX({}^tAAX_0 - {}^tAB) = 0 \\ &\iff {}^tAAX_0 = {}^tAB \end{aligned}$$

Voici le code informatique en langage Python utilisé pour implémenter la méthode d'interpolation et la méthode des moindres carrés, pour stocker les données officielles de l'OMS et pour résoudre numériquement les systèmes différentiels (non linéaires) rencontrés :

```
1  # Construction de la R-algèbre des polynômes
2  def normal(P):
3      try:
4          R=[]
5          i=len(P)-1
6          while P[i]==0:
7              i-=1
8          P=P[:i+1]
9          return P
10     except:  return [0]
11
12 def multi(k,P):
13     return normal([k*a for a in P])
14
15 def somme(P,Q):
```

```
16      P=normal(P)
17      Q=normal(Q)
18      if len(P)!=len(Q):
19          if len(P)>len(Q):
20              while len(Q)!=len(P):
21                  Q.append(0)
22          else:
23              while len(Q)!=len(P):
24                  P.append(0)
25      R=[]
26      for i in range(len(P)):
27          R.append(P[i]+Q[i])
28      return normal(R)
29
30  def Cauchy(P,Q):
```

```
31      p=len(P)
32      s=0
33      for i in range(p):
34          s+=P[i]*Q[p-i-1]
35      return s
36
37
38  def produit(P,Q):
39      P=normal(P)
40      Q=normal(Q)
41      if len(P)!=len(Q):
42          if len(P)>len(Q):
43              while len(Q)!=len(P):
44                  Q.append(0)
45          else:
```

```
46                while len(Q)!=len(P):
47                    P.append(0)
48        p=len(P)
49        R=[]
50        for i in range(p):
51            R.append(Cauchy(P[:i+1],Q[:i+1]))
52        for i in range(p-1):
53            R.append(Cauchy(P[i+1:],Q[i+1:]))
54        return normal(R)
55
56    def Derivation(P):
57        p=len(P)
58        if p==1:
59            return [0]
60        else:
```

```
61            return [i*P[i] for i in range(1,p)]
62
63    def valeur(P,t):
64        s=0
65        for i in range(len(P)):
66            s+=P[i]*(t**i)
67        return s
68
69    # Définition des polynômes de Lagrange et fcts associées
70    def Li(X,i):
71        P=[1]
72        for j in range(len(X)):
73            if j!=i:
74                a=-X[j]/(X[i]-X[j])
75                b=1/(X[i]-X[j])
```

```
76                P=produit(P,[a,b])
77        return normal(P)
78
79    def Lagrange(X,Y):
80        u=len(X)
81        R=[0]
82        for i in range(u):
83            R=somme(R,multi(Y[i],Li(X,i)))
84        return normal(R)
85
86    def poltofct(X,Y):
87        def g(t):
88            return valeur(Lagrange(X,Y),t)
89        return g
90
```

```
91  data=[
92
93  (0,5368,1169),#12
94  (2,6073,1250),#14
95  (7,6190,1267),#19
96  (14,6599,1398),#26
97  (16,6802,1463),#28
98  (18,7312,1583),#30
99  (24,7798,1742),#06
100 (25,7897,1768),#07
101 (27,8014,1857),#09
102 (28,8069,1899),#10
103 (31,8273,2033),#13
104 (32,8356,2085),#14
105 (35,8759,2477),#17
```

```
106    (38,8939,2556),#20
107    (39,9004,2582),#21
108    (42,9203,2655),#24
109    (45,9409,2732),#27
110    (46,9446,2758),#28
111    (49,9633,2827),#31
112    (52,9772,2915),#03
113    (53,9780,2943),#04
114    (55,10030,2977),#06
115    (59,10094,3049),#10
116    (60,10124,3062),#11
117    (61,10150,3067),#12
118    (62,10186,3083),#13
119    (66,10306,3132),#17
120    (67,10340,3145),#18
```

```
121    (68,10362,3153),#19
122    (69,10400,3159),#20
123    (73,10491,3195),#24
124    (74,10518,3199),#25
125    (75,10537,3207),#26
126    (76,10561,3216),#27
127    (81,10740,3276),#1
128    (82,10756,3286),#2
129    (83,10792,3301),#3
130    (88,10934,3341),#8
131    (89,10954,3350),#9
132    (90,10987,3363),#10
133    (95,11103,3408),#15
134    (97,11155,3423),#17
135    (104,11341,3479),#24
```

```
136    (105,11370,3490),#25
137    (111,11466,3546),#1
138    (113,11497,3565),#3
139    (117,11586,3611),#7
140    (119,11619,3629),#9
141    (120,11677,3655),#10
142    (121,11696,3663),#11
143    (124,11742,3687),#14
144    (125,11751,3691),#15
145    (127,11779,3702),#17
146    (131,11829,3742),#21
147    (138,11943,3792),#28
148    (139,11974,3799),#29
149    (146,12139,3832),#05
150    (153,12201,3857),#12
```

```
151    (158,12256,3872),#17
152    (160,12267,3877),#18
153    (163,12294,3885),#21
154    (167,12362,3895),#25
155    (168,12371,3899),#26
156    (170,12287,3901),#28
157    (175,12440,3903),#03
158    (177,12470,3904),#05
159    (181,12519,3904),#09
160    (184,12536,3904),#12
161    (189,12632,3907),#17
162    (195,12696,3908),#23
163    (196,12706,3908),#24
164    (199,12745,3911),#27
165    (202,12816,3911),#30
```

```
166   (209,12884,3913),#06
167   (216,12962,3919),#13
168   (217,12965,3919),#14
169   (220,13012,3926),#17
170   (224,13059,3928),#21
171   (233,13129,3933),#30
172   (238,13155,3940),#05
173   (245,13209,3947),#12
174   (251,13241,3949),#18
175   (254,13262,3949),#21
176   (255,13264,3949),#22
177   (259,13290,3951),#26
178   (262,13387,3951),#29
179   (266,13406,3951),#02
180   (272,13465,3951),#08
```

```
181 (273,13470,3951),#09
182 (278,13489,3952),#14
183 (282,13518,3952),#18
184 (286,13538,3952),#22
185 (287,13541,3952),#23
186 (290,13586,3952),#26
187 (293,13603,3953),#29
188 (296,13638,3953),#01
189 (300,13670,3953),#05
190 (301,13683,3953),#06
191 (303,13697,3953),#08
192 (308,13756,3953),#13
193 (311,13785,3955),#16
194 (314,13811,3955),#19
195 (317,13846,3955),#22
```

```
196     (321,13894,3955),#26
197     (322,13911,3955),#27
198     (329,13945,3955),#04
199     (331,13956,3955),#06
200     (335,13978,3955),#10
201     (336,13982,3955),#11
202     (339,13992,3955),#14
203     (342,13999,3955),#17
204     (349,14052,3955),#24
205     (352,14066,3955),#27
206     (356,14078,3955),#31
207     (363,14122,3955),#07
208
209     ]
210
```

```
211 #reduction nbr de points
212 temps=[i[0] for i in data]
213
214 liste=[(0,5368,1169)]
215 temps2=[0]
216
217 l=len(data)
218 for i in range(l):
219     if data[i][0]-liste[-1][0]>=30:
220         liste.append(data[i])
221         temps2.append(temps[i])
222
223 data = liste
224 temps=temps2
225
```

```
226 #Operateurs sur les fonctions
227 def integrateur(f,a, b, n):
228     integral=0
229     for k in range(n+1):
230         integral+=f(k*(b-a)/n)*(b-a)/n
231     return integral
232
233 def moyenne(f,a,b):
234     return integrateur(f,a,b,10000)/(b-a)
235
236 #Calcul des constantes par interpolartion
237 S0=int(input("Nombre initial de susceptibles : "))
238 S=[S0-i[1] for i in data]
239 I=[i[1]-i[2] for i in data]
240
```

```
241 s=poltofct(temps,S)
242 i=poltofct(temps,I)
243 dsdt=lambda t : derivative(s,t)
244
245 beta = lambda t: -dsdt(t)/(s(t)*i(t))
246 beta= moyenne(beta,0,363)
247
248 #Methode des moindres carres pour SIR
249 S0=int(input("Nombre initial de susceptibles : "))
250 temps=[i[0] for i in data]
251 S=[S0-i[1] for i in data]
252 I=[i[1]-i[2] for i in data]
253
254 plt.plot(temps,S,'b.')
255 plt.plot(temps,I,'b.')
```

```
256 plt.legend(['S'])
257
258 A=[]
259 BS=[]
260 BI=[]
261
262 for i in temps:
263     A.append([1,i,i*i])
264 A=np.array(A)
265 tA=A.transpose()
266
267 for j in S:
268     BS.append([j])
269 for j in I:
270     BI.append([j])
```

```
271
272 BS=np.array(BS)
273 BI=np.array(BI)
274
275 XS=np.dot((A.transpose()),A)
276 XS=np.linalg.inv(XS)
277 XS=np.dot(XS,tA)
278 XS=np.dot(XS,BS)
279 XS=list(XS[0])+list(XS[1])+list(XS[2])
280 dXSdt=Derivation(XS)
281
282 XI=np.dot((A.transpose()),A)
283 XI=np.linalg.inv(XI)
284 XI=np.dot(XI,tA)
285 XI=np.dot(XI,BI)
```

```
286 XI=list(XI[0])+list(XI[1])+list(XI[2])
287 dXIdt=Derivation(XI)
288
289 s = lambda t : valeur(XS,t)
290 i = lambda t : valeur(XI,t)
291 dsdt = lambda t : valeur(dXSdt,t)
292 didt= lambda t : valeur(dXIdt, t)
293
294 beta = lambda t : -(dsdt(t))/(s(t)*i(t))
295 beta = moyenne(beta,0,363)
296
297 gamma= lambda t : beta*s(t)-(didt(t)/i(t))
298 gamma = moyenne(gamma,0, 363)
299
300 #Methode des moindres carres pour SEIR
```

```
301 def plusproche10(d,l):
302     d+=10
303     mini=0
304     while l[mini+1][0]<=d:
305         mini+=1
306     if l[mini+1][0]-d<=d-l[mini][0]:
307         return mini+1
308     else:
309         return mini
310
311 S0=int(input("Nombre initial de susceptibles : "))
312 temps=[i[0] for i in data if i[0]<=353]
313 S=[S0-i[1] for i in data if i[0]<=353]
314 I=[i[1]-i[2] for i in data if i[0]<=353]
315 E=[]
```

```
316 i=0
317 while data[i][0]<=353:
318     E.append(data[plusproche10(data[i][0],data)][1]-data[i]
319     i+=1
320
321 plt.plot(temps,S,'b.')
322 plt.plot(temps,I,'b.')
323 plt.plot(temps,E,'b.')
324 plt.legend(['S'])
325
326 A=[]
327 BS=[]
328 BI=[]
329 BE=[]
330
```

```
331 for i in temps:
332     A.append([1,i,i*i])
333 A=np.array(A)
334 tA=A.transpose()
335
336 for j in S:
337     BS.append([j])
338 for j in I:
339     BI.append([j])
340 for j in E:
341     BE.append([j])
342
343 BS=np.array(BS)
344 BI=np.array(BI)
345 BE=np.array(BE)
```

```
346
347 XS=np.dot((A.transpose()),A)
348 XS=np.linalg.inv(XS)
349 XS=np.dot(XS,tA)
350 XS=np.dot(XS,BS)
351 XS=list(XS[0])+list(XS[1])+list(XS[2])
352 dXSdt=Derivation(XS)
353
354 XI=np.dot((A.transpose()),A)
355 XI=np.linalg.inv(XI)
356 XI=np.dot(XI,tA)
357 XI=np.dot(XI,BI)
358 XI=list(XI[0])+list(XI[1])+list(XI[2])
359 dXIdt=Derivation(XI)
360
```

```
361 XE=np.dot((A.transpose()),A)
362 XE=np.linalg.inv(XE)
363 XE=np.dot(XE,tA)
364 XE=np.dot(XE,BE)
365 XE=list(XE[0])+list(XE[1])+list(XE[2])
366 dXEdt=Derivation(XE)
367
368 s = lambda t : valeur(XS,t)
369 i = lambda t : valeur(XI,t)
370 e = lambda t : valeur(XE,t)
371 dsdt = lambda t : valeur(dXSdt,t)
372 didt= lambda t : valeur(dXIdt, t)
373 dedt= lambda t : valeur(dXEdt, t)
374 beta = lambda t : -(dsdt(t))/(s(t)*i(t))
375 alpha= lambda t : -(dsdt(t)+dedt(t))/e(t)
```

```
376 gamma= lambda t: -(dsdt(t)+dedt(t)+didt(t))/i(t)
377 beta = moyenne(beta,0,353)
378 alpha=moyenne(alpha,0,353)
379 gamma=moyenne(gamma,0,353)
380
381 import numpy as np
382 from scipy.integrate import odeint
383 import matplotlib.pyplot as plt
384
385 #Resolution du SDNL POUR SIR
386 def f(s,t,b,G,alpha):
387     S=s[0]
388     I=s[1]
389     R=s[2]
390     dSdt=-b*S*I
```

```
391         dIdt=b*S*I - (G+alpha)*I
392         dRdt=G*I
393         return [dSdt,dIdt,dRdt]
394
395     t=np.linspace(0,360,1000000)
396     s0=[92837,5368,0]
397
398     s=odeint(f,s0,t, (2.7141037173344333e-08,0.0007573689040734
399
400     plt.plot(t,s[:,0],'r--', linewidth=2.0)
401     plt.plot(t,s[:,1],'r--', linewidth=2.0)
402     plt.plot(t,s[:,2],'b.', linewidth=2.0)
403     plt.xlabel("t")
404     plt.ylabel("S")
405     plt.legend(["S"])
```

```
406 plt.show()
407
408 #RESOLUTION DU SDNL POUR SEIR
409 def f(s,t,b,G,alpha,nu):
410     S=s[0]
411     E=s[1]
412     I=s[2]
413     R=s[3]
414     dSdt=-b*S*I
415     dEdt=b*S*I-alpha*E
416     dIdt=alpha*E - (G+nu)*I
417     dRdt=G*I
418     return [dSdt,dEdt,dIdt,dRdt]
419
420 t=np.linspace(0,353,100000)
```

```
421 s0=[92837,822,4199,0]
422
423 s=odeint(f,s0,t,(3.049348123160306e-08,0.0011394041241835893
424
425 plt.plot(t,s[:,0],'r--', linewidth=2.0)
426 plt.plot(t,s[:,1],'b-', linewidth=2.0)
427 plt.plot(t,s[:,2],'b', linewidth=2.0)
428 plt.plot(t,s[:,3],'y.', linewidth=2.0)
429 plt.xlabel("t")
430 plt.ylabel("S[S,I]")
431 plt.legend(["S","I"])
432 plt.show()
```