

Modélisation des tumeurs cérébrales

Partie : Codes Python

UE Stage

Laura FUENTES VICENTE
- Année 2021 -



IJCLAB – CNRS

Adresse

Maître de Stage : Mathilde BADOUAL

Enseignant Référent : Sabir JACQUIR

Directeur du laboratoire : Achille STOCCHI

Table des matières

1	Modélisation sans Radiothérapie - Méthode Matrice	3
1.1	La diffusion, la prolifération et la comparaison à la solution exacte :	3
1.2	Modélisation du processus de diffusion-prolifération - Méthode numerical Recipes :	7
2	Application de la Radiothérapie :	12
2.1	Processus de diffusion prolifération suite à une session de RT	12
2.2	Étude du paramètre M	18
2.2.1	Variation du Rayon tumoral pour plusieurs valeurs de M	18
2.2.2	Étude de la pente associée au rayon tumoral	25
2.2.3	Variations des intervalles ΔG et Δg et valeurs du rayon tumoral minimal	31
2.3	Étude du paramètre p :	35
2.3.1	Variations du rayon tumoral pour plusieurs valeurs de p	35
2.3.2	Étude de la pente associée à la courbe du rayon tumoral	42
2.3.3	Variations des intervalles ΔG , Δg et des valeurs du rayon tumoral . .	48

Pour retourner vers la partie principale du rapport de stage, faire click **ici** `run:/path/to/my/file.ext`

1 Modélisation sans Radiothérapie - Méthode Matrice

1.1 La diffusion, la prolifération et la comparaison à la solution exacte :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #définition des paramètres du temps
5 Tempstot = 40 #en années
6 Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
7 ##de 10**-2 années par itération
8 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
9 print("nbiterations_=", int(nb_iterations))
10
11
12 #paramètre L= taille physique boite(mm)/
13 L= 45 #en mm
14 dx= 0.01 #en mm
15 dx2 = (dx)**2 #en mm**2
16 N1 = int(L/dx) #taille de la representation de l'espace
17 print("N1=", N1)
18 X = np.arange(0, L, dx)
19 print("X=", X)
20
21 D = 1 #coeff de diffusion mm^2/ans
22 alpha = (D*(Delta_t)/dx2) #defini par rapport à l'intervalle de temps, dx et
    D
23 k=1 #coeff de prolifération (en ans^-1)
24 print("alpha=", alpha)
25
26 #création des figures vides
27 fig, ax1 = plt.subplots(ncols=1)
28 fig.subplots_adjust(wspace=0.75)
29 fig2, ax2 = plt.subplots(ncols=1)
30 fig2.subplots_adjust(wspace=0.75)
31 fig3, ax3 = plt.subplots(ncols=1)
32 fig3.subplots_adjust(wspace=0.75)
33
34 #matrice M1D pour les prochains calculs
35 def MatriceM1(N1, alpha):
36     MD1 = np.zeros((N1,N1))
37
38     for i in range (0, N1):
39         for j in range (0, N1):
40             if (i == j) :
41                 MD1[i,j]= 1. + 2*alpha
42             elif (i+1 == j or i-1 == j) :
43                 MD1[i,j]= -alpha
44     MD1[0,1]= 0. -2*alpha
45     MiD1= np.linalg.inv(MD1)
```

```

46     return MD1 , MiD1
47
48 MD1, MiD1 = MatriceM1(N1, alpha)
49 print("MD1=␣", MD1)
50
51 #Matrice avec valeurs initiales de U0 à partir
52 #des conditions initiales données par la gaussienne
53 def MatriceU0(N1, X, t0):
54     U0=np.zeros(N1)
55     for i in range (0, N1-1):
56         U0[i] = np.exp(-(X[i]**2)/(4*D*t0))
57     return U0
58
59 U0 = MatriceU0(N1, X, 0.1)
60
61 #définition de la solution exacte de la diffusion pour comparer
62
63 def sol_exacteD(X, t0, t):
64     sol_exacteD=np.zeros(N1)
65     for i in range(0, N1-1):
66         sol_exacteD[i] =
67             ((t0/(t-t0))*(1/2))*(np.exp( -(X[i]**2) / (4*D*(t+t0))))
68     return sol_exacteD
69 sol_exacteD = sol_exacteD(X, 0.1, Tempstot)
70
71 #création d'un vecteur vide que l'on remplira par la suite
72 Tpreuve = []
73
74 def ResD( Tpreuve):
75     UT=np.zeros(N1)
76     Tpreuve.extend([U0])
77     UT=U0
78     U=np.zeros(N1)
79     for i in range(0, nb_iterations-1):
80         U= np.dot(MiD1, UT)
81         UT = U
82         Tpreuve.extend([U])
83     return Tpreuve
84 Tpreuve = ResD(Tpreuve)
85 print(np.shape(Tpreuve))
86
87
88 #définition de la solution exacte de la diffusion+prolif pour comparer
89 def sol_exacteT(X, t0, t):
90     sol_exacteT=np.zeros(N1)
91     for i in range(0, N1-1):
92         sol_exacteT[i] = ((t0/(t+t0))*(1/2)) * (np.exp( -(X[i]**2) / (4*D*(
93             t+t0)))) * (np.exp(k*t) )
94     return sol_exacteT
95
96 sol_exacteDP= sol_exacteT(X, 0.1, Tempstot)

```

```

96
97 def U6(N1, t0):
98     U6=np.zeros(N1)
99     for i in range (0, N1-1):
100         U6[i] = np.exp(-(X[i]**2 + 1.5)/(4*D*t0))
101     return U6
102
103 U6 = U6(N1, 0.1)
104 #Calcul prolifération une itération)
105 def Prolif(U6):
106     UP= np.zeros(N1)
107     for i in range(0, N1-1):
108         UP[i] = U6[i] + k*Delta_t*U6[i]*(1- (U6[i]))
109     U6=UP
110     return UP
111
112 UN=[]
113 #on repète n fois la fonction de la prolifération
114 def nfois(nb_iterations, U6):
115     UN.extend([U6])
116     for i in range(0, nb_iterations-1):
117         UY = Prolif(U6)
118         U6=UY
119         UN.extend([UY])
120     return UN
121 UN = nfois(nb_iterations, U6)
122
123 #définition des localisation dans les figures
124 left, width = .9, .2
125 bottom, height = .1, 1.5
126 right = left + width
127 top = bottom + height
128
129 #représentation graphique
130 ax1.set_title("DIFFUSION")
131 ax1.set_ylabel("Densité_cellulaire")
132 ax1.set_xlabel("Longueur_de_la_boîte_(en_μmm)")
133 ax1.set_xlim([0,10])
134 ax1.text(1.1, -0.3, "D=1.0_μmm^2/an",
135         horizontalalignment='right',
136         verticalalignment='top',
137         transform=ax1.transAxes)
138 ax1.plot(X, Tpreuve[0], label="Sol_approch_diff_t=0*dt")
139 ax1.plot(X, Tpreuve[5], label="Sol_approch_diff_t=5*dt")
140 ax1.plot(X, Tpreuve[20], label="Sol_approch_diff_t=20*dt")
141 ax1.plot(X, Tpreuve[200], label="Sol_approch_diff_t=200*dt")
142 ax1.plot(X, Tpreuve[500], label="Sol_approch_diff_t=500*dt")
143 ax1.plot(X, Tpreuve[1000], label="Sol_approch_diff_t=1000*dt")
144 ax1.plot(X, Tpreuve[2500], label="Sol_approch_diff_t=2500*dt")
145 ax1.plot(X, Tpreuve[nb_iterations-1], label="Sol_ex_diff_t=Tempstot")
146 ax1.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower_left')

```

```

147
148 ax2.set_title("PROLIFÉRATION")
149 ax2.set_ylabel("Densité_cellulaire")
150 ax2.set_xlabel("Longueur_de_la_boîte_(en_mm)")
151 ax2.set_xlim([0,10])
152 ax2.text(1.1, -0.3, "K=1_an^-1",
153         horizontalalignment='right',
154         verticalalignment='top',
155         transform=ax2.transAxes)
156 ax2.plot(X, UN[0], label="_Sol_ex_prolif_t=8*dt")
157 ax2.plot(X, UN[10], label="_Sol_ex_prolif_t=100*dt")
158 ax2.plot(X, UN[200], label="_Sol_ex_prolif_t=200*dt")
159 ax2.plot(X, UN[500], label="_Sol_ex_prolif_t=500*dt")
160 ax2.plot(X, UN[1000], label="_Sol_ex_prolif_t=1000*dt")
161 ax2.plot(X, UN[2500], label="_Sol_ex_prolif_t=2500*dt")
162 ax2.plot(X, UN[3500], label="_Sol_ex_prolif_t=3500*dt")
163 ax2.plot(X, UN[nb_iterations-1], label="_Sol_ex_prolif_t=nb_iterations*dt")
164 ax2.legend(bbox_to_anchor=(-0.7, -0.7, -0.7, -0.7), loc='lower_left')
165
166 ax3.set_title("COMPARAISON_SOLUTION_APPROCHÉE_SOLUTION_EXACTE")
167 ax3.set_ylabel("Densité_cellulaire")
168 ax3.set_xlabel("Longueur_de_la_boîte_(en_mm)")
169 ax3.text(1.1, -0.3, "D=1mm^2/an",
170         horizontalalignment='right',
171         verticalalignment='top',
172         transform=ax3.transAxes)
173 ax3.plot(X, sol_exacteD, label="_Sol_ex_diff_t=tempstot")
174 ax3.plot(X, Tpreuve[nb_iterations-1], label="_Sol_approch_diff_t=tempstot")
175 ax3.legend(bbox_to_anchor=(-0.3, -0.3, -0.3, -0.3), loc='lower_left')
176
177 #affichage des figures
178 fig.show()
179 fig2.show()
180 fig3.show()

```

1.2 Modélisation du processus de diffusion-prolifération - Méthode numerical Recipes :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #paramètres généraux du temps
5 Tempstot = 30 #en années
6 Delta_t = 0.01 ##en années, il s'agit du pas, on avance
7 ##en effet de 10**-2 années par itération
8 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
9 print("nbiterations= ", int(nb_iterations))
10
11 #paramètres généraux de l'espace et autres
12 #calculer L= taille physique boite(mm)/
13 L= 70 #en mm
14 dx= 0.01 #en mm
15 dx2 = (dx)**2
16 N1 = int(L/dx) #taille de la représentation
17 X = np.arange(0 , L, dx)
18 D = 1 #coeff de diffusion (en mm^2/ans)
19 k=1 #coeff de prolifération (en ans^-1)
20 alpha = (D*Delta_t)/dx2 #défini par rapport à l'intervalle de temps, dx et D
21 seuil = np.full(N1, 0.07)
22 seuill = 0.07 #valeur du seuil de detection
23 vitfront = 2*np.sqrt(D*k) #vitesse de front 2sqrt(Dk)
24 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k)) #vitesse de front + 10%
25 vit_front=np.full(nb_iterations-1, 2*np.sqrt(D*k)) #fonction constante avec
    la vitesse de front
26 vit_frontdix=np.full(nb_iterations-1, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
    #fonction contante avec la vitesse de front + 10%
27
28 #matrice M1D pour la modélisation
29 def MatriceMD1(N1, alpha):
30     MD1 = np.zeros((N1,N1))
31     for i in range (0, N1):
32         for j in range (0, N1):
33             if (i == j) :
34                 MD1[i,j]= 1. + 2*alpha
35             elif (i+1 == j or i-1 == j) :
36                 MD1[i,j]= -alpha
37     MD1[0,1]= MD1[0,1]-alpha
38     return MD1
39
40 MD1 = MatriceMD1(N1, alpha)
41 print("MD1= ", MD1)
42
43 #on stocke les diagonales dans les nouveaux vecteurs vides
44 A=np.full(N1, 1+2*alpha)
45 C=np.full(N1, -alpha)
46 C[0]=0
```

```

47 B=np.full(N1, -alpha)
48 B[0]=-2*alpha
49 B[N1-1]=0
50
51 #Matrice avec conditions initiales U0 données par la marche
52 U01=np.zeros(N1)
53 U01[0]=1
54 U01[1]=1
55
56 #calcul de la diffusion une seule itération
57 def resol(B, C, A, R ):
58     gam = np.zeros(N1)
59     U=np.zeros(N1)
60     if (B[0]==0):
61         print("ERROR!")
62     bet=B[0]
63     U[0]= (R[0]/(bet))
64     for j in range(1, N1-1):
65         gam[j] = C[j-1]/bet
66         bet=B[j]-A[j]*gam[j]
67         if (bet==0):
68             print("ERROR!")
69         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
70     k=N1-2
71     while k>=0:
72         U[k] -= gam[k+1]*U[k+1]
73         k = k-1
74     gam=np.zeros(N1)
75     return U
76
77 #boucle pour calculer la diffusion au rang n+1
78 def solutions de la diffusion (MD1, N1, f):
79     if(f==0):
80         return U01
81     U = np.zeros(N1)
82     UI = np.zeros(N1)
83     UI = resol(A, B, C, U01)
84     if (f==1):
85         return UI
86     for i in range(2,nb_iterations+3):
87         U = resol(A, B, C, UI)
88         UI = U
89         if i==(f):
90             return U
91
92 solutionsadt= solutions(MD1, N1, nb_iterations-1)
93
94 #def d'un tableau vide que l'on remplira avec les valeurs de la diffprolif a
95     tout temps fixé
96 TpreuveTemp=[]
97 #fonction calculant les valeurs de la diffusion puis de la prolifération

```



```

97
98 def DiffProlif(TpreuveTemp, U01):
99     oo=np.zeros(N1)
100     for m in range(0, N1-1):
101         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
102
103     TpreuveTemp.extend([oo])
104     for i in range(1, nb_iterations-1):
105         TP= np.zeros(N1)
106         UP=np.zeros(N1)
107         TP = resol(A,B,C, TpreuveTemp[i-1])
108         for j in range(0, N1-1):
109             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
110         TpreuveTemp.extend([UP])
111     return TpreuveTemp
112
113 TpreuveTemp = DiffProlif(TpreuveTemp, U01) #remplissement du vecteur
diffprolif
114
115 #Calcul du Rayon
116 RX= []
117 RY = []
118 def rayon(TpreuveTemp, seuill, RX, RY):
119     for i in range(0, nb_iterations-1):
120         r=0
121         for j in range(1, N1-1):
122             if ( ( TpreuveTemp[i][j] <= seuill ) and ( TpreuveTemp[i][j-1] >=
seuill ) ):
123                 r = (((j-1)+j)/2)*dx
124             RX.extend([r])
125             RY.extend([i*Delta_t])
126     return RX, RY
127
128 RX, RY = rayon(TpreuveTemp, seuill, RX, RY)
129
130 #calcul de la pente associée au rayon tumoral
131 def pente(RY, RX):
132     pente = np.zeros(nb_iterations-1)
133     for i in range(4, nb_iterations-5):
134         pente[i] = (RX[i+4] - RX[i-4]) / (RY[i+4] - RY[i-4])
135     return pente
136
137 pente = pente(RY, RX)
138 max_value = max(pente)
139 maxi = (np.where(pente == max_value)[0][0])*Delta_t
140 print("maxi", maxi)
141
142 #calcul du tlim
143 def vitfrontpente(pente, vitfront):
144     e=0
145     vit10 = []

```

```

146     g=0
147     vit0=[]
148     for i in range(0, nb_iterations-1):
149         if ( ( pente[i] <= vitfront ) and ( pente[i-1] >= vitfront ) ):
150             g= (((i-1)+i)/2)*Delta_t
151             vit0.extend([g])
152
153         elif(( pente[i] <= vitfrontdix ) and ( pente[i-1] >= vitfrontdix )):
154             e= (((i-1)+i)/2)*Delta_t
155             vit10.extend([e])
156
157     return vit0, vit10
158 g, e = vitfrontpente(pente, vitfront)
159
160 #création des figures vides
161 fig1, ax1 = plt.subplots(ncols=1)
162 fig1.subplots_adjust(wspace=0.75)
163 fig2, ax2 = plt.subplots(ncols=1)
164 fig2.subplots_adjust(wspace=0.75)
165 fig3, ax3 = plt.subplots(ncols=1)
166 fig3.subplots_adjust(wspace=0.75)
167
168 #définition des localisations sur les figures
169 left, width = .9, .2
170 bottom, height = .1, 1.5
171 right = left + width
172 top = bottom + height
173
174 #représentation graphique
175 ax1.set_title("DIFFUSION_PROLIFERATION_SANS_RT")
176 ax1.set_xlabel("Longueur_de_la_boîte_(en_mm)")
177 ax1.set_ylabel("Densité_cellulaire")
178 ax1.text(.8, -0.4, "D=1mm^2/an_K=1ans^-1",
179         horizontalalignment='right',
180         verticalalignment='top',
181         transform=ax1.transAxes)
182 ax1.plot(X, TpreuveTemp[0], label="Sol_approch_t=0dt")
183 ax1.plot(X, TpreuveTemp[20], label="Sol_approch_t=20dt")
184 ax1.plot(X, TpreuveTemp[75], label="Sol_approch_t=75dt")
185 ax1.plot(X, TpreuveTemp[250], label="Sol_approch_t=250dt")
186 ax1.plot(X, TpreuveTemp[750], label="Sol_approch_t=750dt")
187 ax1.plot(X, TpreuveTemp[1500], label="Sol_approch_t=1500dt")
188 ax1.plot(X, TpreuveTemp[2500], label="Sol_approch_t=2500dt")
189 ax1.plot(X, TpreuveTemp[nb_iterations-2], label="Sol_approch_t=Tempstot")
190 ax1.plot(X, seuil, label="Seuil=0.07mm")
191 ax1.legend(bbox_to_anchor=(-0.7, -0.7, -0.7, -0.7), loc='lower_left')
192
193 ax2.set_title("VARIATION_DU_RAYON_TUMORAL")
194 ax2.set_xlabel("Temps_(en_années)")
195 ax2.set_ylabel("Rayon_de_la_tumeur_(en_mm)")
196 ax2.text(.8, -0.4, "D=1mm^2/an_K=1ans^-1",

```

```

197         horizontalalignment='right',
198         verticalalignment='top',
199         transform=ax2.transAxes)
200 ax2.set_ylim([0,L])
201 ax2.plot(RY, RX)
202 ax2.legend()
203
204 ax3.set_title("VARIATION DE LA PENTE")
205 ax3.grid(True)
206 ax3.set_xlabel("Temps (en années)")
207 ax3.set_ylabel("Coefficient de la pente")
208 ax3.text(.8, -0.4, "D=1mm^2/an K=1ans^-1",
209         horizontalalignment='right',
210         verticalalignment='top',
211         transform=ax3.transAxes)
212 ax3.set_xlim([maxi-0.5,Tempstot-1])
213 ax3.set_ylim([0,10])
214 ax3.annotate('t0=4.20 années, t10%=3.75 années', xy=(4.8,2.1), xytext
215             =(6,7.4), arrowprops={'facecolor':'black', 'shrink':0.05} )
216 ax3.plot(RY, pente, label="Variation de la pente en fonction du temps")
217 ax3.plot(RY, vit_front, label=r'$f(x)=\sqrt{Dk}$')
218 ax3.plot(RY, vit_frontdix, label=r'$g(x)=2\sqrt{Dk}$+10%')
219 ax3.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lowerleft')
220
221 #affichage figures
222 fig1.show()
223 fig2.show()
224 fig3.show()

```

2 Application de la Radiothérapie :

2.1 Processus de diffusion prolifération suite à une session de RT

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 \\
4 Tempstot = 30 #en années
5 Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
6 #de 10**-2 années par itération
7 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
8 print("nbiterations_=", int(nb_iterations))
9
10 left, width = .9, .2
11 bottom, height = .1, 1.5
12 right = left + width
13 top = bottom + height
14
15
16 #paramètres pour l'espace et autres
17 #calculer L= taille physique boîte (mm)/
18 L= 70 en mm
19 dx= 0.01 #en mm
20 dx2 = (dx)**2
21 N1 = int(L/dx) #taille de la representation de l'espace
22 X = np.arange(0 , L, dx)
23 D = 1 #coeff de diffusion
24 k=1 #coeff de prolifération
25 P15=0.9 #proportion des cellules atteintes par la RT
26 M=2.5 #taux de mort des cellules atteintes par la RT (en ans^-1)
27 nbt0 = 950 #Moment de l'appli de la RT
28 alpha = (D*Delta_t)/dx2 #defini par rapport à l'intervalle de temps et dx
29 print(alpha)
30 seuil = np.full(N1, 0.07)
31 seuill = 0.07
32 vitfront = 2*np.sqrt(D*k)
33 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
34 vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
35 vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
36
37
38 #matrice M1D pour les prochains calculs
39 def MatriceMD1(N1, alpha):
40     MD1 = np.zeros((N1,N1))
41     for i in range (0, N1):
42         for j in range (0, N1):
43             if (i == j) :
44                 MD1[i,j]= 1. + 2*alpha
45             elif (i+1 == j or i-1 == j) :
46                 MD1[i,j]= -alpha
47     MD1[0,1]= MD1[0,1]-alpha
```

```

48     return MD1
49
50 MD1 = MatriceMD1(N1, alpha)
51 print("MD1=␣", MD1)
52
53 #on stocke les diagonales dans les nouveaux vecteurs vides
54 A=np.full(N1, 1+2*alpha)
55 C=np.full(N1, -alpha)
56 C[0]=0
57 B=np.full(N1, -alpha)
58 B[0]=-2*alpha
59 B[N1-1]=0
60
61 #vecteur avec les conditions initiales données par la marche
62 U01=np.zeros(N1)
63 U01[0]=1
64 U01[1]=1
65
66 #fonction diffusion une seule itération
67 def resol(B, C, A, R ):
68     gam = np.zeros(N1)
69     U=np.zeros(N1)
70     if (B[0]==0):
71         print("ERROR!")
72     bet=B[0]
73     U[0]= (R[0]/(bet))
74     for j in range(1, N1-1):
75         gam[j] = C[j-1]/bet
76         bet=B[j]-A[j]*gam[j]
77         if (bet==0):
78             print("ERROR!")
79         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
80     k=N1-2
81     while k>=0:
82         U[k] -= gam[k+1]*U[k+1]
83         k = k-1
84     gam=np.zeros(N1)
85     return U
86
87 #boucle pour calculer au rang n+1 les solutions de la diffusion
88 def solutions(MD1, N1, f):
89     if(f==0):
90         return U01
91     U = np.zeros(N1)
92     UI = np.zeros(N1)
93     UI = resol(A, B, C, U01)
94     if (f==1):
95         return UI
96     for i in range(2,nb_iterations+3):
97         U = resol(A, B, C, UI)
98         UI = U

```

```

99         if i==(f+1):
100             return U
101
102 solutionsadt= solutions(MD1, N1, nb_iterations)
103
104 #fonction de la diffusion prolifération avant RT
105 TpreuveTempRT15=[]
106 TpreuveTempn15=[]
107 sommem15 = []
108
109 zero = np.zeros(N1)
110 def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
111     oo=np.zeros(N1)
112     for m in range(0, N1-1):
113         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
114     sommem.extend([oo])
115     TpreuveTempn.extend([oo])
116     TpreuveTempRT.extend([zero])
117     for i in range(1, nbt0):
118         TP= np.zeros(N1)
119         UP=np.zeros(N1)
120         TP = resol(A,B,C, TpreuveTempn[i-1])
121         for j in range(0, N1-1):
122             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
123         sommem.extend([UP])
124         TpreuveTempn.extend([UP])
125         TpreuveTempRT.extend([zero])
126     return TpreuveTempn, TpreuveTempRT, sommem
127
128 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
129     TpreuveTempRT15, U01, nbt0, sommem15) #remplissage du vecteur
130     diffprolif
131
132 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
133     ès RT
134 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
135     oo=np.zeros(N1)
136     op=np.zeros(N1)
137     for i in range(0, N1-1):
138         oo[i]= p*TpreuveTempn[nbt0-1][i]
139         op[i]= (1-p)*TpreuveTempn[nbt0-1][i]
140
141     Tpreuver = resol(A,B,C, oo)
142     Tpreuven = resol(A,B,C, op)
143     oo = np.zeros(N1)
144     op = np.zeros(N1)
145     s = np.zeros(N1)
146     for m in range(0, N1-1):
147         oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))

```

```

146         op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
147         s[m]= op[m] + oo[m]
148
149     sommem.extend([s])
150     TpreuveTempRT.extend([oo])
151     TpreuveTempn.extend([op])
152     for i in range(nbt0+1, nb_iterations):
153
154         TPRT= np.zeros(N1)
155         UPRT=np.zeros(N1)
156         TPRT = resol(A,B,C, TpreuveTempRT[i-1])
157         s = np.zeros(N1)
158         TP= np.zeros(N1)
159         UP=np.zeros(N1)
160         TP = resol(A,B,C, TpreuveTempn[i-1])
161         for j in range(0, N1-1):
162             UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
163             cd= TPRT[j]
164             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
165             s[j] = UP[j] +UPRT[j]
166
167         sommem.extend([s])
168         TpreuveTempRT.extend([UPRT])
169         TpreuveTempn.extend([UP])
170     return TpreuveTempn, TpreuveTempRT, sommem
171
172
173 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M,
    TpreuveTempRT15, nbt0, sommem15, P15)
174
175
176 #Calcul du rayon
177
178 RXn15= []
179 RYn15 = []
180
181 def rayonn(TpreuveTempn, seuill, RXn, RYn):
182     for i in range(0, nb_iterations):
183         r=0
184         for j in range(1, N1-1):
185             if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
                >= seuill ) ):
186                 r = (((j-1)+j)/2)*dx
187             RXn.extend([r])
188             RYn.extend([i*Delta_t])
189     return RXn, RYn
190
191 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
192

```

```

193 #Calcul de la valeur minimale du rayon et deltag (intervalle de temps pour
    atteindre cette dernière)
194 def minn(RX):
195     minim=35
196     indice = 0
197     deltag=0
198     for i in range (nbt0, nb_iterations):
199         m = RX[i]
200         if (m<minim):
201             minim = m
202             indice = i *Delta_t
203             deltag= (i*Delta_t) - (nbt0*Delta_t)
204     return minim, indice, deltag
205
206
207 minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
208 print("minimRX", minimRX15)
209 print("indice", indicemin15)
210 print("deltag", deltag15)
211
212 #calcul de l'intervalle de temps DeltaG
213 def momentegal(RX):
214     bingo=RX[nbt0]
215     DELTAG=0
216     for i in range(nbt0, nb_iterations):
217         if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
218             DELTAG = (i*Delta_t) - (nbt0*Delta_t)
219     return DELTAG
220
221
222 DELTAG15 = momentegal(RXn15)
223 print("DELTAG15= ",DELTAG15)
224
225 #création des figures vides
226 fig2, ax2 = plt.subplots(ncols=1)
227 fig2.subplots_adjust(wspace=0.75)
228
229 fig4, ax4 = plt.subplots(ncols=1)
230 fig4.subplots_adjust(wspace=0.75)
231
232 #représentation graphique
233
234 ax2.set_title("_DIFFUSION_PROLIFERATION_CELLULES_")
235 ax2.grid(True)
236 ax2.text(.8, -0.4, "D=1mm^2/an_K=1_ans^-1_P=0.9_M=2.5_ans^-1",
237         horizontalalignment='right',
238         verticalalignment='top',
239         transform=ax2.transAxes)
240 ax2.set_xlabel("Longueur_de_la_boîte_(en_mm)")

```



```

241 ax2.set_ylabel("Densité_cellulaire")
242 ax2.plot(X, sommem15[0], label="sol_approch_c_t=0dt", color='gray' )
243 ax2.plot(X, sommem15[20], label="sol_approch_c_t=20dt", color='gray')
244 ax2.plot(X, sommem15[50], label="sol_approch_c_t=50dt", color='gray')
245 ax2.plot(X, sommem15[250], label="sol_approch_c_t=250dt", color='gray')
246 ax2.plot(X, sommem15[400], label="sol_approch_c_t=400dt", color='gray')
247 ax2.plot(X, sommem15[550], label="sol_approch_c_t=550dt", color='gray')
248 ax2.plot(X, sommem15[625], label="sol_approch_c_t=625dt", color='gray')
249 ax2.plot(X, sommem15[750], label="sol_approch_c_t=750dt", color='gray')
250 ax2.plot(X, sommem15[875], label="sol_approch_c_t=875dt", color='gray')
251 ax2.plot(X, sommem15[950], label="sol_approch_c_moment_irradiation_RT_t=950
    dt", color='red')
252 ax2.plot(X, sommem15[1000], label="sol_approch_c_t=1000dt", color='orange')
253 ax2.plot(X, sommem15[1150], label="sol_approch_c_t=1150dt", color='pink')
254 ax2.plot(X, sommem15[1300], label="sol_approch_c_t=1300dt", color='magenta'
    )
255 ax2.plot(X, sommem15[1500], label="sol_approch_c_t=1500dt", color='purple')
256 ax2.plot(X, sommem15[1750], label="sol_approch_c_t=1750dt", color='blue')
257 ax2.plot(X, sommem15[2000], label="sol_approch_c_t=2000dt", color='green')
258 ax2.plot(X, sommem15[2250], label="sol_approch_c_t=2250dt", color='black')
259 ax2.plot(X, sommem15[2250], label="sol_approch_c_t=2750dt", color='black')
260 ax2.plot(X, sommem15[nb_iterations-1], label="sol_approch_c_t=Tempstot",
    color='black')
261 ax2.legend(bbox_to_anchor=(-0.99, -0.99, -0.99, -0.99), loc='lower_left')
262
263
264 ax4.set_title("VARIATION_DU_RAYON_TUMORAL")
265 ax4.grid(True)
266 ax4.set_xlabel("Temps_(en_années)")
267 ax4.set_ylabel("Rayon_de_la_tumeur_(en_mm)")
268 ax4.text(.8, -0.4, "D=1mm^2/an_K=1ans^-1_M=1.1ans^-1_fixé",
269         horizontalalignment='right',
270         verticalalignment='top',
271         transform=ax4.transAxes)
272 ax4.annotate('Session_de_radiothérapie', xy=(9,7.1), xytext=(2.5,17.2),
    arrowprops={'facecolor':'black', 'shrink':0.05} )
273 ax4.plot(RYn15, RXn15, label="P=0.9")
274 ax4.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower_left')
275
276 #affichage figures
277 fig2.show()
278 fig4.show()

```

2.2 Étude du paramètre M

2.2.1 Variation du Rayon tumoral pour plusieurs valeurs de M

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #définition des paramètres temps
5 Tempstot = 20 #en années
6 Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
7 #de 10**-2 années par itération
8 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
9 print("nbiterations_=", int(nb_iterations))
10
11 #paramètres de l'espace et autres
12 #calculer L= taille physique boite(mm)/
13 L= 70 #en mm
14 dx= 0.01 #en mm
15 dx2 = (dx)**2
16 N1 = int(L/dx) #taille de la représentation de l'espace
17 X = np.arange(0 , L, dx)
18 D = 1 #coeff de diffusion
19 k=1 #coeff de prolifération
20 M03=1.5 #taux de mort
21 M06=2.5 #taux de mort
22 M15=4 #taux de mort
23 p=0.95 #proportion de cellules atteintes par la RT
24 #proportion des cellules atteintes par la RT
25 nbt0 = 950 #moment de l'appli de la RT
26 alpha = (D*Delta_t)/dx2 #defini par rapport à l'intervalle de temps, dx et D
27 print(alpha)
28 seuil = np.full(N1, 0.07)
29 seuill = 0.07 #seuil de détection
30 vitfront = 2*np.sqrt(D*k)
31 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
32 vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
33 vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
34
35
36 #matrice M1D pour les prochains calculs
37 def MatriceMD1(N1, alpha):
38     MD1 = np.zeros((N1,N1))
39     for i in range (0, N1):
40         for j in range (0, N1):
41             if (i == j) :
42                 MD1[i,j]= 1. + 2*alpha
43             elif (i+1 == j or i-1 == j) :
44                 MD1[i,j]= -alpha
45     MD1[0,1]= MD1[0,1]-alpha
46     return MD1
47
48 MD1 = MatriceMD1(N1, alpha)
```

```

49 print("MD1=␣", MD1)
50
51 #on stocke les diagonales dans les nouveaux vecteurs vides
52 A=np.full(N1, 1+2*alpha)
53 C=np.full(N1, -alpha)
54 C[0]=0
55 B=np.full(N1, -alpha)
56 B[0]=-2*alpha
57 B[N1-1]=0
58
59 #conditions initiales données par la marche
60 U01=np.zeros(N1)
61 U01[0]=1
62 U01[1]=1
63
64 #fonction diffusion pour une seule itération
65 def resol(B, C, A, R ):
66     gam = np.zeros(N1)
67     U=np.zeros(N1)
68     if (B[0]==0):
69         print("ERROR!")
70     bet=B[0]
71     U[0]= (R[0]/(bet))
72     for j in range(1, N1-1):
73         gam[j] = C[j-1]/bet
74         bet=B[j]-A[j]*gam[j]
75         if (bet==0):
76             print("ERROR!")
77         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
78     k=N1-2
79     while k>=0:
80         U[k] -= gam[k+1]*U[k+1]
81         k = k-1
82     gam=np.zeros(N1)
83     return U
84
85 #boucle pour calculer la diffusion au rang n+1
86 def solutions(MD1, N1, f):
87     if(f==0):
88         return U01
89     U = np.zeros(N1)
90     UI = np.zeros(N1)
91     UI = resol(A, B, C, U01)
92     if (f==1):
93         return UI
94     for i in range(2,nb_iterations+3):
95         U = resol(A, B, C, UI)
96         UI = U
97         if i==(f+1):
98             return U
99

```

```

100 solutionsadt= solutions(MD1, N1, nb_iterations)
101
102 #création des vecteurs vides que l'on utilisera par la suite:
103 TpreuveTempRT03=[]
104 TpreuveTempn03=[]
105 sommem03 = []
106
107 TpreuveTempRT06=[]
108 TpreuveTempn06=[]
109 sommem06 = []
110
111 TpreuveTempRT15=[]
112 TpreuveTempn15=[]
113 sommem15 = []
114
115 zero = np.zeros(N1)
116 #modélisation de la diffusion prolifération avant RT
117 def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
118     oo=np.zeros(N1)
119     for m in range(0, N1-1):
120         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
121     sommem.extend([oo])
122     TpreuveTempn.extend([oo])
123     TpreuveTempRT.extend([zero])
124     for i in range(1, nbt0):
125         TP= np.zeros(N1)
126         UP=np.zeros(N1)
127         TP = resol(A,B,C, TpreuveTempn[i-1])
128         for j in range(0, N1-1):
129             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
130         sommem.extend([UP])
131         TpreuveTempn.extend([UP])
132         TpreuveTempRT.extend([zero])
133     return TpreuveTempn, TpreuveTempRT, sommem
134 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
135     TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
136     diffprolif
137 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
138     TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
139     diffprolif
140 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
141     TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
142     diffprolif
143
144 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
145     ès RT
146
147 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem):
148     oo=np.zeros(N1)
149     op=np.zeros(N1)

```

```

144     for i in range(0, N1-1):
145         oo[i]= p*TpreuveTempn[nbt0-1][i]
146         op[i]= (1-p)*TpreuveTempn[nbt0-1][i]
147
148     Tpreuver = resol(A,B,C, oo)
149     Tpreuven = resol(A,B,C, op)
150     oo = np.zeros(N1)
151     op = np.zeros(N1)
152     s = np.zeros(N1)
153     for m in range(0, N1-1):
154         oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
155         op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
156         s[m]= op[m] + oo[m]
157
158     sommem.extend([s])
159     TpreuveTempRT.extend([oo])
160     TpreuveTempn.extend([op])
161     for i in range(nbt0+1, nb_iterations):
162
163         TPRT= np.zeros(N1)
164         UPRT=np.zeros(N1)
165         TPRT = resol(A,B,C, TpreuveTempRT[i-1])
166         s = np.zeros(N1)
167         TP= np.zeros(N1)
168         UP=np.zeros(N1)
169         TP = resol(A,B,C, TpreuveTempn[i-1])
170         for j in range(0, N1-1):
171             UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
172             cd= TPRT[j]
173             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
174             s[j] = UP[j] +UPRT[j]
175
176         sommem.extend([s])
177         TpreuveTempRT.extend([UPRT])
178         TpreuveTempn.extend([UP])
179     return TpreuveTempn, TpreuveTempRT, sommem
180
181 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M03
    , TpreuveTempRT03, nbt0, sommem03)
182 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M06
    , TpreuveTempRT06, nbt0, sommem06)
183 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M15
    , TpreuveTempRT15, nbt0, sommem15)
184
185
186 #Calcul du rayon
187 RXn03= []
188 RYn03 = []
189
190 RXn06= []

```

```

191 RYn06 = []
192
193 RXn15= []
194 RYn15 = []
195
196 def rayonn(TpreuveTempn, seuill, RXn, RYn):
197     for i in range(0, nb_iterations):
198         r=0
199         for j in range(1, N1-1):
200             if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
201                 >= seuill ) ):
202                 r = (((j-1)+j)/2)*dx
203             RXn.extend([r])
204             RYn.extend([i*Delta_t])
205         return RXn, RYn
206
207 RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
208 RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
209 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
210
211 #On cherche a trouver le moment où le rayon est minimal suite a la session
212 #de RT (deltag) et la valeur minimale du rayon
213
214 def minn(RX):
215     minim=35
216     indice = 0
217     deltag=0
218     for i in range (nbt0, nb_iterations):
219         m = RX[i]
220         if (m<minim):
221             minim = m
222             indice = i *Delta_t
223             deltag= (i*Delta_t) - (nbt0*Delta_t)
224     return minim, indice, deltag
225
226 minimRX03, indicemin03, deltag03 = minn(RXn03) #fonction retournant la
227 #valeur minimum du rayon ainsi que le moment m auquel cette valeur min
228 #est atteinte
229 print("minimRX", minimRX03)
230 print("indice", indicemin03)
231 print("deltag", deltag03)
232
233 minimRX06, indicemin06, deltag06 = minn(RXn06) #fonction retournant la
234 #valeur minimum du rayon ainsi que le moment m auquel cette valeur min
235 #est atteinte
236 print("minimRX", minimRX06)
237 print("indice", indicemin06)
238 print("deltag", deltag06)
239
240 minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la
241 #valeur minimum du rayon ainsi que le moment m auquel cette valeur min

```

```

    est atteinte
235 print("minimRX", minimRX15)
236 print("indice", indicemin15)
237 print("deltag", deltag15)
238
239 #calcul de l'intervalle DeltaG
240 def momentegal(RX):
241     bingo=RX[nbt0]
242     DELTAG=0
243     for i in range(nbt0, nb_iterations):
244         if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
245             DELTAG = (i*Delta_t) - (nbt0*Delta_t)
246     return DELTAG
247
248 DELTAG03 = momentegal(RXn03)
249 print("DELTAG03_=",DELTAG03)
250
251 DELTAG06 = momentegal(RXn06)
252 print("DELTAG06_=",DELTAG06)
253
254 DELTAG15 = momentegal(RXn15)
255 print("DELTAG15_=",DELTAG15)
256
257 #création des figures vides
258 fig4, ax4 = plt.subplots(ncols=1)
259 fig4.subplots_adjust(wspace=0.75)
260
261 #localisation dans les figures
262 left, width = .9, .2
263 bottom, height = .1, 1.5
264 right = left + width
265 top = bottom + height
266
267 #représentation graphique
268 ax4.set_title("VARIATION_DU_RAYON_TUMORAL_")
269 ax4.grid(True)
270 ax4.set_xlim([0, 18])
271 ax4.set_ylim([0, 25])
272 ax4.set_xlabel("Temps_(en_années)")
273 ax4.set_ylabel("Rayon_de_la_tumeur_(en_mm)")
274 ax4.text(.8, -0.4, "D=1mm^2/an_K=1_ans^-1_P=0.95_fixés_RT=9.5_ans",
275         horizontalalignment='right',
276         verticalalignment='top',
277         transform=ax4.transAxes)
278 ax4.annotate('Session_de_radiothérapie', xy=(9.5,15.1), xytext=(3.5,20.2),
279            arrowprops={'facecolor':'black', 'shrink':0.05} )
280 ax4.plot(RYn03, RXn03, label="M=1.1")
281 ax4.plot(RYn06, RXn06, label="M=2.5")
282 ax4.plot(RYn15, RXn15, label="M=4")
283 ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower_left')

```

```
284 #affichage des figures
285 fig4.show()
```


2.2.2 Étude de la pente associée au rayon tumoral

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 #paramètres du temps
6 Tempstot = 20 #en années
7 Delta_t = 0.01 ##en années, il s'agit du pas, on avance
8 #en effet de 10**-2 années par itération
9 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
10 print("nbiterations_=", int(nb_iterations))
11
12 #paramètres de l'espace et autres
13 #calculer L= taille physique boîte(mm)/
14 L= 70 #en mm
15 dx= 0.01 #en mm
16 dx2 = (dx)**2
17 N1 = int(L/dx) #taille de la représentation de l'espace
18 X = np.arange(0 , L, dx)
19 D = 1 #coeff de diffusion
20 k=1 #coeff de prolifération
21 P=0.95 #proportion de cellules atteintes par la RT
22 M15=1.1 #taux de mort
23 M25=2.5 #taux de mort
24 M4=4 #taux de mort
25 nbt0 = 950 #appli de la RT
26 alpha = (D*Delta_t)/dx2 #défini par rapport à l'intervalle de temps, dx et D
27 print(alpha)
28 seuil = np.full(N1, 0.07)
29 seuill = 0.07 #seuil de détection
30 vitfront = 2*np.sqrt(D*k)
31 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
32 vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
33 vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
34
35
36 #matrice M1D pour les prochains calculs
37 def MatriceMD1(N1, alpha):
38     MD1 = np.zeros((N1,N1))
39     for i in range (0, N1):
40         for j in range (0, N1):
41             if (i == j) :
42                 MD1[i,j]= 1. + 2*alpha
43             elif (i+1 == j or i-1 == j) :
44                 MD1[i,j]= -alpha
45     MD1[0,1]= MD1[0,1]-alpha
46     return MD1
47
48 MD1 = MatriceMD1(N1, alpha)
49 print("MD1=", MD1)
```

```

50
51 #on stocke les diagonales dans les nouveaux vecteurs vides
52 A=np.full(N1, 1+2*alpha)
53 C=np.full(N1, -alpha)
54 C[0]=0
55 B=np.full(N1, -alpha)
56 B[0]=-2*alpha
57 B[N1-1]=0
58
59 #définition d'un vecteur avec les conditions initiales données par la marche
60 U01=np.zeros(N1)
61 U01[0]=1
62 U01[1]=1
63
64 #résolution diffusion pour une seule itération
65 def resol(B, C, A, R ):
66     gam = np.zeros(N1)
67     U=np.zeros(N1)
68     if (B[0]==0):
69         print("ERROR!")
70     bet=B[0]
71     U[0]= (R[0]/(bet))
72     for j in range(1, N1-1):
73         gam[j] = C[j-1]/bet
74         bet=B[j]-A[j]*gam[j]
75         if (bet==0):
76             print("ERROR!")
77         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
78     k=N1-2
79     while k>=0:
80         U[k] -= gam[k+1]*U[k+1]
81         k = k-1
82     gam=np.zeros(N1)
83     return U
84
85 #boucle pour calculer la diffusion au rang n+1
86 def solutions(MD1, N1, f):
87     if(f==0):
88         return U01
89     U = np.zeros(N1)
90     UI = np.zeros(N1)
91     UI = resol(A, B, C, U01)
92     if (f==1):
93         return UI
94     for i in range(2,nb_iterations+3):
95         U = resol(A, B, C, UI)
96         UI = U
97         if i==(f+1):
98             return U
99
100 solutionsadt= solutions(MD1, N1, nb_iterations)

```

```

101
102 #création des vecteurs vides pour la suite
103 TpreuveTempRT03=[]
104 TpreuveTempn03=[]
105 sommem03 = []
106
107 TpreuveTempRT06=[]
108 TpreuveTempn06=[]
109 sommem06 = []
110
111 TpreuveTempRT15=[]
112 TpreuveTempn15=[]
113 sommem15 = []
114
115 zero = np.zeros(N1)
116 #modélisation de la diffusion prolifération avant RT
117 def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
118     oo=np.zeros(N1)
119     for m in range(0, N1-1):
120         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
121     sommem.extend([oo])
122     TpreuveTempn.extend([oo])
123     TpreuveTempRT.extend([zero])
124     for i in range(1, nbt0):
125         TP= np.zeros(N1)
126         UP=np.zeros(N1)
127         TP = resol(A,B,C, TpreuveTempn[i-1])
128         for j in range(0, N1-1):
129             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
130         sommem.extend([UP])
131         TpreuveTempn.extend([UP])
132         TpreuveTempRT.extend([zero])
133     return TpreuveTempn, TpreuveTempRT, sommem
134 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
135     TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
136     diffprolif
137
138 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
139     TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
140     diffprolif
141
142 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
143     TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
144     diffprolif
145
146
147 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
148     ès RT
149 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
150     oo=np.zeros(N1)
151     op=np.zeros(N1)
152     for i in range(0, N1-1):
153         oo[i]= p*TpreuveTempn[nbt0-1][i]

```

```

145         op[i]= (1-p)*TpreuveTempn[nbt0-1][i]
146
147     Tpreuver = resol(A,B,C, oo)
148     Tpreuven = resol(A,B,C, op)
149     oo = np.zeros(N1)
150     op = np.zeros(N1)
151     s = np.zeros(N1)
152     for m in range(0, N1-1):
153         oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
154         op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
155             Tpreuver[m] )
156         s[m]= op[m] + oo[m]
157
158     sommem.extend([s])
159     TpreuveTempRT.extend([oo])
160     TpreuveTempn.extend([op])
161     for i in range(nbt0+1, nb_iterations):
162
163         TPRT= np.zeros(N1)
164         UPRT=np.zeros(N1)
165         TPRT = resol(A,B,C, TpreuveTempRT[i-1])
166         s = np.zeros(N1)
167         TP= np.zeros(N1)
168         UP=np.zeros(N1)
169         TP = resol(A,B,C, TpreuveTempn[i-1])
170         for j in range(0, N1-1):
171             UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
172             cd= TPRT[j]
173             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
174             s[j] = UP[j] +UPRT[j]
175
176         sommem.extend([s])
177         TpreuveTempRT.extend([UPRT])
178         TpreuveTempn.extend([UP])
179     return TpreuveTempn, TpreuveTempRT, sommem
180
181 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M15
182     , TpreuveTempRT03, nbt0, sommem03, P)
183
184 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M25
185     , TpreuveTempRT06, nbt0, sommem06, P)
186
187 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M4,
188     TpreuveTempRT15, nbt0, sommem15, P)
189
190 #Calcul du rayon
191 RXn03= []
192 RYn03 = []
193
194 RXn06= []
195 RYn06 = []

```

```

192 RXn15= []
193 RYn15 = []
194
195 def rayonn(TpreuveTempn, seuill, RXn, RYn):
196     for i in range(0, nb_iterations):
197         r=0
198         for j in range(1, N1-1):
199             if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
200                 >= seuill ) ):
201                 r = (((j-1)+j)/2)*dx
202                 RXn.extend([r])
203                 RYn.extend([i*Delta_t])
204             return RXn, RYn
205
206 RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
207 RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
208 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
209
210 #calcul de la pente associée à la courbe du rayon
211 def pente(RY, RX):
212     pente = np.zeros(nb_iterations)
213     for i in range(4, nb_iterations-5):
214         pente[i] = (RX[i+4] - RX[i-4]) / (RY[i+4] - RY[i-4])
215     return pente
216
217 pente03 = pente(RYn03, RXn03)
218 max_value03 = max(pente03)
219 maxi03 = (np.where(pente03 == max_value03)[0][0])*Delta_t
220 print("maxi03", maxi03)
221
222 pente06 = pente(RYn06, RXn06)
223 pente06 = pente(RYn06, RXn06)
224 max_value06 = max(pente06)
225 maxi06 = (np.where(pente06 == max_value06)[0][0])*Delta_t
226 print("maxi06", maxi06)
227
228 pente15 = pente(RYn15, RXn15)
229 pente15 = pente(RYn15, RXn15)
230 max_value15 = max(pente15)
231 maxi15 = (np.where(pente15 == max_value15)[0][0])*Delta_t
232 print("maxi15", maxi15)
233
234 #création des figures vides
235 fig4, ax4 = plt.subplots(ncols=1)
236 fig4.subplots_adjust(wspace=0.75)
237
238 #localisations sur les figures
239 left, width = .9, .2
240 bottom, height = .1, 1.5
241 right = left + width
242 top = bottom + height

```

```

242 #représentation graphique
243 ax4.set_title("VARIATION DE LA PENTE")
244 ax4.grid(True)
245 ax4.set_xlabel("Temps (en années)")
246 ax4.set_xlim([0, 19])
247 ax4.set_ylabel("Coefficient de la pente")
248 ax4.text(.8, -0.4, "D=1mm^2/an, K=1 ans^-1 fixées",
249           horizontalalignment='right',
250           verticalalignment='top',
251           transform=ax4.transAxes)
252 ax4.plot(RYn03, pente03, label="Pour M=1.1 pour P=0.95")
253 ax4.plot(RYn06, pente06, label="Pour M=2.5 pour P=0.95")
254 ax4.plot(RYn15, pente15, label="Pour M=4 pour P=0.95")
255 ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower left')
256
257 #affichage des figures
258 fig4.show()

```

2.2.3 Variations des intervalles ΔG et Δg et valeurs du rayon tumoral minimal

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #valeurs qu'on a calculé auparavant
5 RXmp06 = [0.6, 0.7, 0.8, 0.9, 0.99, 1.5, 2.5, 3.0, 4.0]
6 RXmp09= [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 1.5, 2.5, 3.0, 4.0]
7
8 #pour P=0.6
9 VARRAYMINP06 = [6.165, 6.165,6.165, 6.165, 6.165, 6.105, 5.885, 5.775,
10 5.595]
11 VARDELTATP06 = [0.0005,0.001,0.05, 0.07, 0.1, 0.48, 0.74,0.79, 0.83]
12 VARdeltatP06= [0, 0, 0,0, 0, 0.17, 0.31, 0.35, 0.34]
13
14 #pour P=0.7
15 VARRAYMINP07 = [6.165, 6.165,6.165, 6.145, 6.115, 5.935, 5.565, 5.415,
16 5.155]
17 VARDELTATP07 = [0.0005,0.03,0.2, 0.42, 0.53, 0.88, 1.07,1.1, 1.12]
18 VARdeltatP07= [0, 0, 0,0.07, 0.23, 0.42, 0.5, 0.46, 0.45]
19
20 #pour P=0.8
21 VARRAYMINP08 = [6.145, 6.105,6.055, 5.995, 5.935, 5.585, 4.995, 4.755,
22 4.365]
23 VARDELTATP08 = [0.43,0.71,0.9, 1.05, 1.14, 1.38, 1.49,1.5, 1.51]
24 VARdeltatP08= [0.06, 0.31, 0.4,0.47, 0.53, 0.65, 0.65, 0.64, 0.59]
25
26 #pour P=0.9
27 VARRAYMINP09seul = [ 6.155,6.155 ,6.135 , 6.045,5.925, 5.785,5.645, 5.495,
28 5.365, 4.665, 3.525, 3.025, 2.085]
29 VARDELTATP09seul = [0.05,0.1,0.75 , 1.32, 1.6,1.77,1.9, 1.97, 2.02, 2.14,
30 2.17,2.17, 2.16]
31 VARdeltatP09seul= [0, 0, 0.28,0.73 , 0.86, 1.02, 1.02, 1.11, 1.11, 1.11,
32 0.97, 0.92, 0.82]
33
34 #valeurs calculées auparavant
35 VARRAYMINP09 = [5.925, 5.785,5.645, 5.495, 5.365, 4.665, 3.525, 3.025,
36 2.085]
37 VARDELTATP09 = [1.6,1.77,1.9, 1.97, 2.02, 2.14, 2.17,2.17, 2.16]
38 VARdeltatP09= [0.86, 1.02, 1.02, 1.11, 1.11, 1.11, 0.97, 0.92, 0.82]
39
40 #création des figures vides
41 fig1, ax1 = plt.subplots()
42 figa, axa = plt.subplots()
43 fig2, ax2 = plt.subplots()
44 figb, axb = plt.subplots()
45 fig3, ax3 = plt.subplots()
46 figc, axc = plt.subplots()
47 fig4, ax4 = plt.subplots()
48 figd, axd = plt.subplots()
```

```

43 fig5, ax5 = plt.subplots()
44 fig6, ax6 = plt.subplots()
45 fig7, ax7 = plt.subplots()
46
47 #représentation graphique
48 ax1.set_title("Variations de DELTAT et deltat en fonction des valeurs de M
    pour P=0.6 fixé")
49 ax1.set_xlabel("Valeurs de M")
50 ax1.grid(True)
51 ax1.set_ylabel("Intervalle de temps en années")
52 ax1.plot(RXmp06, VARDELATP06, label="Variation de DeltaT")
53 ax1.plot(RXmp06, VARdeltatP06, label="Variation de deltat")
54 ax1.legend()
55
56 #affichage des figures
57 fig1.show()
58
59 #représentation graphique
60 axa.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de M pour P=0.6 fixé")
61 axa.set_xlabel("Valeurs de M")
62 axa.grid(True)
63 axa.set_ylabel("Rayon minimal (en mm)")
64 axa.plot(RXmp06, VARRAYMINP06)
65
66 #affichage figures
67 figa.show()
68
69 #représentation graphique
70 ax2.set_title("Variations de DELTAT et deltat en fonction des valeurs de M
    pour P=0.7 fixé")
71 ax2.set_xlabel("Valeurs de M")
72 ax2.grid(True)
73 ax2.set_ylabel("Intervalle de temps en années")
74 ax2.plot(RXmp06, VARDELATP07, label="Variation de DeltaT")
75 ax2.plot(RXmp06, VARdeltatP07, label="Variation de deltat")
76 ax2.legend()
77
78 #affichage des figures
79 fig2.show()
80
81 #représentation graphique
82 axb.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de M pour P=0.7 fixé")
83 axb.set_xlabel("Valeurs de M")
84 axb.grid(True)
85 axb.set_ylabel("Rayon minimal (en mm)")
86 axb.plot(RXmp06, VARRAYMINP07)
87
88 #affichage des figures
89 figb.show()

```



```

90
91 #représentation graphique
92 ax3.set_title("Variations de DELTAT et deltat en fonction des valeurs de M
    pour P=0.8 fixé")
93 ax3.set_xlabel("Valeurs de M")
94 ax3.grid(True)
95 ax3.set_ylabel("Intervalle de temps en années")
96 ax3.plot(RXmp06, VARDELATP08, label="Variation de DeltaT")
97 ax3.plot(RXmp06, VARdeltatP08, label="Variation de deltat")
98 ax3.legend()
99
100 #affichage des figures
101 fig3.show()
102
103 #représentation graphique
104 axc.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de M pour P=0.8 fixé")
105 axc.set_xlabel("Valeurs de M")
106 axc.grid(True)
107 axc.set_ylabel("Rayon minimal (en mm)")
108 axc.plot(RXmp06, VARRAYMINP08)
109
110 #affichage des figures
111 figc.show()
112
113
114 #représentation graphique
115 ax4.set_title("Variations de DELTAT et deltat en fonction des valeurs de M
    pour P=0.9 fixé")
116 ax4.set_xlabel("Valeurs de M")
117 ax4.grid(True)
118 ax4.set_ylabel("Intervalle de temps en années")
119 ax4.plot(RXmp09, VARDELATP09seul, label="Variation de DeltaT")
120 ax4.plot(RXmp09, VARdeltatP09seul, label="Variation de deltat")
121 ax4.legend()
122
123 #affichage des figures
124 fig4.show()
125
126 #représentation graphique
127 axd.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de M pour P=0.9 fixé")
128 axd.set_xlabel("Valeurs de M")
129 axd.grid(True)
130 axd.set_ylabel("Rayon min (en mm)")
131 axd.plot(RXmp09, VARRAYMINP09seul)
132 figd.show()
133 figd.savefig("NRVARRAYONPFIXE09.pdf", bbox_inches='tight')
134
135 #affichage des figures

```

```

136 ax5.set_title("Variations de DELTAT en fonction des valeurs de M pour
    plusieurs valeurs de p fixées")
137 ax5.set_xlabel("Valeurs de M")
138 ax5.set_ylabel("Intervalle de temps en années")
139 ax5.grid(True)
140 ax5.plot(RXmp06, VARDELATP06, label="Variation de Delta T pour P=0.6 fixé")
141 ax5.plot(RXmp06, VARDELATP07, label="Variation de Delta T pour P=0.7 fixé")
142 ax5.plot(RXmp06, VARDELATP08, label="Variation de Delta T pour P=0.8 fixé")
143 ax5.plot(RXmp06, VARDELATP09, label="Variation de Delta T pour P=0.9 fixé")
144 ax5.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
145 fig5.show()
146
147 #représentation graphique
148 ax6.set_title("Variations de deltat en fonction des valeurs de M pour
    plusieurs valeurs de p fixées")
149 ax6.set_xlabel("Valeurs de M")
150 ax6.set_ylabel("Intervalle de temps en années")
151 ax6.grid(True)
152 ax6.plot(RXmp06, VARdeltatP06, label="Variation de deltat pour P=0.6 fixé")
153 ax6.plot(RXmp06, VARdeltatP07, label="Variation de deltat pour P=0.7 fixé")
154 ax6.plot(RXmp06, VARdeltatP08, label="Variation de deltat pour P=0.8 fixé")
155 ax6.plot(RXmp06, VARdeltatP09, label="Variation de deltat pour P=0.9 fixé")
156 ax6.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
157
158 #affichage des figures
159 fig6.show()
160
161 #représentation graphique
162 ax7.set_title("Variations du rayon en fonction des valeurs de M pour
    plusieurs valeurs de P fixées")
163 ax7.set_xlabel("Valeurs de M")
164 ax7.set_ylabel("Rayon tumoral (en mm)")
165 ax7.grid(True)
166 ax7.plot(RXmp06, VARRAYMINP06, label="Variation de rayon pour P=0.6 fixé")
167 ax7.plot(RXmp06, VARRAYMINP07, label="Variation de rayon pour P=0.7 fixé")
168 ax7.plot(RXmp06, VARRAYMINP08, label="Variation de rayon pour P=0.8 fixé")
169 ax7.plot(RXmp06, VARRAYMINP09, label="Variation de rayon pour P=0.9 fixé")
170 ax7.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
171
172 #affichage des figures
173 fig7.show()

```

2.3 Étude du paramètre p :

2.3.1 Variations du rayon tumoral pour plusieurs valeurs de p

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #paramètres du temps
5 Tempstot = 20 #en années
6 Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
7 #de 10**-2 années par itération
8 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
9 print("nbiterations_=", int(nb_iterations))
10
11 #localisations sur les figures
12 left, width = .9, .2
13 bottom, height = .1, 1.5
14 right = left + width
15 top = bottom + height
16
17
18 #paramètres de l'espace et autres
19 #calculer L= taille physique boîte(mm)/
20 L= 70 #en mm
21 dx= 0.01 #en mm
22 dx2 = (dx)**2
23 N1 = int(L/dx) #taille de la représentation de l'espace
24 X = np.arange(0 , L, dx)
25 D = 1 #coeff de diffusion
26 k=1 #coeff de prolifération
27 P03=0.65 #proportion de cellules atteintes par la RT
28 P06=0.85 #proportion de cellules atteintes par la RT
29 P15=0.95 #proportion de cellules atteintes par la RT
30 M=2.5 #taux de mort
31 nbt0 = 950 #appli de la RT
32 alpha = (D*Delta_t)/dx2 #défini par rapport à l'intervalle de temps, dx et D
33 print(alpha)
34 seuil = np.full(N1, 0.07)
35 seuill = 0.07 #seuil de détection
36 vitfront = 2*np.sqrt(D*k)
37 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
38 vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
39 vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
40
41
42 #matrice M1D pour les prochains calculs
43 def MatriceMD1(N1, alpha):
44     MD1 = np.zeros((N1,N1))
45     for i in range (0, N1):
46         for j in range (0, N1):
47             if (i == j) :
48                 MD1[i,j]= 1. + 2*alpha
```

```

49         elif (i+1 == j or i-1 == j) :
50             MD1[i,j]= -alpha
51     MD1[0,1]= MD1[0,1]-alpha
52     return MD1
53
54 MD1 = MatriceMD1(N1, alpha)
55 print("MD1=␣", MD1)
56
57 #on stocke les diagonales dans les nouveaux vecteurs vides
58 A=np.full(N1, 1+2*alpha)
59 C=np.full(N1, -alpha)
60 C[0]=0
61 B=np.full(N1, -alpha)
62 B[0]=-2*alpha
63 B[N1-1]=0
64
65 #conditions initiales données par la marche
66 U01=np.zeros(N1)
67 U01[0]=1
68 U01[1]=1
69
70 #calcul d'une seule itération pour la diffusion
71 def resol(B, C, A, R ):
72     gam = np.zeros(N1)
73     U=np.zeros(N1)
74     if (B[0]==0):
75         print("ERROR!")
76     bet=B[0]
77     U[0]= (R[0]/(bet))
78     for j in range(1, N1-1):
79         gam[j] = C[j-1]/bet
80         bet=B[j]-A[j]*gam[j]
81         if (bet==0):
82             print("ERROR!")
83         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
84     k=N1-2
85     while k>=0:
86         U[k] -= gam[k+1]*U[k+1]
87         k = k-1
88     gam=np.zeros(N1)
89     return U
90
91 #boucle pour calculer la diffusion au rang n+1
92 def solutions(MD1, N1, f):
93     if(f==0):
94         return U01
95     U = np.zeros(N1)
96     UI = np.zeros(N1)
97     UI = resol(A, B, C, U01)
98     if (f==1):
99         return UI

```

```

100     for i in range(2,nb_iterations+3):
101         U = resol(A, B, C, UI)
102         UI = U
103         if i==(f+1):
104             return U
105
106 solutionsadt= solutions(MD1, N1, nb_iterations)
107
108 #création des vecteurs vides pour la suite
109
110 TpreuveTempRT03=[]
111 TpreuveTempn03=[]
112 sommem03 = []
113
114 TpreuveTempRT06=[]
115 TpreuveTempn06=[]
116 sommem06 = []
117
118 TpreuveTempRT15=[]
119 TpreuveTempn15=[]
120 sommem15 = []
121
122 zero = np.zeros(N1)
123 #fonction calculant la diffusion prolifération avant la RT
124 def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
125     oo=np.zeros(N1)
126     for m in range(0, N1-1):
127         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
128     sommem.extend([oo])
129     TpreuveTempn.extend([oo])
130     TpreuveTempRT.extend([zero])
131     for i in range(1, nbt0):
132         TP= np.zeros(N1)
133         UP=np.zeros(N1)
134         TP = resol(A,B,C, TpreuveTempn[i-1])
135         for j in range(0, N1-1):
136             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
137         sommem.extend([UP])
138         TpreuveTempn.extend([UP])
139         TpreuveTempRT.extend([zero])
140     return TpreuveTempn, TpreuveTempRT, sommem
141 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
142     TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
143     diffprolif
144 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
145     TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
146     diffprolif
147 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
148     TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
149     diffprolif

```

```

145
146 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
    ès RT
147 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
148     oo=np.zeros(N1)
149     op=np.zeros(N1)
150     for i in range(0, N1-1):
151         oo[i]= p*TpreuveTempn[nbt0-1][i]
152         op[i]= (1-p)*TpreuveTempn[nbt0-1][i]
153
154     Tpreuver = resol(A,B,C, oo)
155     Tpreuven = resol(A,B,C, op)
156     oo = np.zeros(N1)
157     op = np.zeros(N1)
158     s = np.zeros(N1)
159     for m in range(0, N1-1):
160         oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
161         op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
162         s[m]= op[m] + oo[m]
163
164     sommem.extend([s])
165     TpreuveTempRT.extend([oo])
166     TpreuveTempn.extend([op])
167     for i in range(nbt0+1, nb_iterations):
168
169         TPRT= np.zeros(N1)
170         UPRT=np.zeros(N1)
171         TPRT = resol(A,B,C, TpreuveTempRT[i-1])
172         s = np.zeros(N1)
173         TP= np.zeros(N1)
174         UP=np.zeros(N1)
175         TP = resol(A,B,C, TpreuveTempn[i-1])
176         for j in range(0, N1-1):
177             UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
178             cd= TPRT[j]
179             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
180             s[j] = UP[j] +UPRT[j]
181
182         sommem.extend([s])
183         TpreuveTempRT.extend([UPRT])
184         TpreuveTempn.extend([UP])
185     return TpreuveTempn, TpreuveTempRT, sommem
186
187 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M,
    TpreuveTempRT03, nbt0, sommem03, P03)
188 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M,
    TpreuveTempRT06, nbt0, sommem06, P06)
189 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M,
    TpreuveTempRT15, nbt0, sommem15, P15)
190

```

```

191
192 #Calcul du rayon tumoral
193 RXn03= []
194 RYn03 = []
195
196 RXn06= []
197 RYn06 = []
198
199 RXn15= []
200 RYn15 = []
201
202 def rayonn(TpreuveTempn, seuill, RXn, RYn):
203     for i in range(0, nb_iterations):
204         r=0
205         for j in range(1, N1-1):
206             if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
207                 >= seuill ) ):
208                 r = (((j-1)+j)/2)*dx
209             RXn.extend([r])
210             RYn.extend([i*Delta_t])
211         return RXn, RYn
212
213 RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
214 RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
215 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
216
217 #On cherche a trouver le moment où le rayon est minimal suite a la session
218 #de RT (deltag) et sa valeur
219
220 def minn(RX):
221     minim=35
222     indice = 0
223     deltag=0
224     for i in range (nbt0, nb_iterations):
225         m = RX[i]
226         if (m<minim):
227             minim = m
228             indice = i *Delta_t
229             deltag= (i*Delta_t) - (nbt0*Delta_t)
230     return minim, indice, deltag
231
232 minimRX03, indicemin03, deltag03 = minn(RXn03) #fonction retournant la
233 #valeur minimum du rayon ainsi que le moment m auquel cette valeur min
234 #est atteinte
235
236 print("minimRX", minimRX03)
237 print("indice", indicemin03)
238 print("deltag", deltag03)
239
240
241
242 minimRX06, indicemin06, deltag06 = minn(RXn06) #fonction retournant la
243 #valeur minimum du rayon ainsi que le moment m auquel cette valeur min
244 #est atteinte

```

```

236 print("minimRX", minimRX06)
237 print("indice", indicemin06)
238 print("deltag", deltag06)
239
240 minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
241 print("minimRX", minimRX15)
242 print("indice", indicemin15)
243 print("deltag", deltag15)
244
245 #étude du DeltaG
246 def momentegal(RX):
247     bingo=RX[nbt0]
248     DELTAG=0
249     for i in range(nbt0, nb_iterations):
250         if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
251             DELTAG = (i*Delta_t) - (nbt0*Delta_t)
252     return DELTAG
253
254 DELTAG03 = momentegal(RXn03)
255 print("DELTAG03_=", DELTAG03)
256
257 DELTAG06 = momentegal(RXn06)
258 print("DELTAG06_=", DELTAG06)
259
260 DELTAG15 = momentegal(RXn15)
261 print("DELTAG15_=", DELTAG15)
262
263 #création des figures vides
264 fig2, ax2 = plt.subplots(ncols=1)
265 fig2.subplots_adjust(wspace=0.75)
266
267 fig4, ax4 = plt.subplots(ncols=1)
268 fig4.subplots_adjust(wspace=0.75)
269
270 #représentation graphique
271 ax2.set_title("_DIFFUSION_PROLIFERATION_CELLULES_")
272 ax2.grid(True)
273 ax2.text(.8, -0.4, "D=1mm^2/an_K=1ans^-1_P=0.9M=1.1ans^-1",
274         horizontalalignment='right',
275         verticalalignment='top',
276         transform=ax2.transAxes)
277 ax2.set_xlabel("Longueur_de_la_boîte_(en_mm)")
278 ax2.set_ylabel("Densité_cellulaire")
279 ax2.plot(X, sommem03[0], label="sol_approch_ t=0dt", color='black' )
280 ax2.plot(X, sommem03[20], label="sol_approch_ t=20dt", color='black')
281 ax2.plot(X, sommem03[50], label="sol_approch_ t=50dt", color='black')
282 ax2.plot(X, sommem03[250], label="sol_approch_ t=250dt", color='black')
283 ax2.plot(X, sommem03[500], label="sol_approch_ t=500dt_moment_irradiation_RT
    ", color='red')

```



```

284 ax2.plot(X, sommem03[550], label="sol_approch_Δt=550dt", color='yellow')
285 ax2.plot(X, sommem03[625], label="sol_approch_Δt=625dt", color='orange')
286 ax2.plot(X, sommem03[750], label="sol_approch_Δt=750dt", color='magenta')
287 ax2.plot(X, sommem03[850], label="sol_approch_Δt=850dt", color='purple')
288 ax2.plot(X, sommem03[950], label="sol_approch_Δt=950dt", color='blue')
289 ax2.plot(X, sommem03[1150], label="sol_approch_Δt=1150dt", color='green')
290 ax2.plot(X, sommem03[1500], label="sol_approch_Δt=1500dt", color='gray')
291 ax2.plot(X, sommem03[nb_iterations-1], label="sol_approch_Δt=Tempstot",
        color='gray')
292 ax2.legend(bbox_to_anchor=(-0.9, -0.9, -0.9, -0.9), loc='lower_left')
293
294
295 ax4.set_title("VARIATION DU RAYON TUMORAL")
296 ax4.grid(True)
297 ax4.set_xlim([0, 18])
298 ax4.set_ylim([0, 35])
299 ax4.set_xlabel("Temps (en années)")
300 ax4.set_ylabel("Rayon de la tumeur (en mm)")
301 ax4.text(.8, -0.4, "D=1mm2/an K=1 an-1 M=2.5 an-1 fixés",
302         horizontalalignment='right',
303         verticalalignment='top',
304         transform=ax4.transAxes)
305 ax4.annotate('Session de radiothérapie', xy=(9.5,15.1), xytext=(3.5,20.2),
        arrowprops={'facecolor':'black', 'shrink':0.05} )
306 ax4.plot(RYn03, RXn03, label="P=0.65")
307 ax4.plot(RYn06, RXn06, label="P=0.85")
308 ax4.plot(RYn15, RXn15, label="P=0.95")
309 ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower_left')
310
311 #affichage des figures
312
313 fig2.show()
314 fig4.show()

```

2.3.2 Étude de la pente associée à la courbe du rayon tumoral

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #paramètres du temps
5 Tempstot = 20 #en années
6 Delta_t = 0.01 ##en années, il s'agit du pas, on avance
7 #en effet de 10**-2 années par itération
8 nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
9 print("nbiterations_=", int(nb_iterations))
10
11 #paramètres de l'espace et autres
12 #calculer L= taille physique boite (mm)/
13 L= 70 #en mm
14 dx= 0.01 #en mm
15 dx2 = (dx)**2
16 N1 = int(L/dx) #taille de la représentation de l'espace
17 X = np.arange(0 , L, dx)
18 D = 1 #coeff de diffusion
19 k=1 #coeff de prolifération
20 P03=0.65 #proportion des cellules atteintes par la RT
21 P06=0.85 #proportion des cellules atteintes par la RT
22 P15=0.95 #proportion des cellules atteintes par la RT
23 M=2.5 #taux de mort
24 #proportion des cellules atteintes par la RT
25 nbt0 = 950 #Appli de la RT
26 alpha = (D*Delta_t)/dx2 #defini par rapport à l'intervalle de temps, dx et D
27 print(alpha)
28 seuil = np.full(N1, 0.07)
29 seuill = 0.07 #seuil de détection
30 vitfront = 2*np.sqrt(D*k)
31 vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
32 vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
33 vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))
34
35
36 #matrice M1D pour les prochains calculs
37 def MatriceMD1(N1, alpha):
38     MD1 = np.zeros((N1,N1))
39     for i in range (0, N1):
40         for j in range (0, N1):
41             if (i == j) :
42                 MD1[i,j]= 1. + 2*alpha
43             elif (i+1 == j or i-1 == j) :
44                 MD1[i,j]= -alpha
45     MD1[0,1]= MD1[0,1]-alpha
46     return MD1
47
48 MD1 = MatriceMD1(N1, alpha)
49 print("MD1=", MD1)
```

```

50
51 #on stocke les diagonales dans les nouveaux vecteurs vides
52 A=np.full(N1, 1+2*alpha)
53 C=np.full(N1, -alpha)
54 C[0]=0
55 B=np.full(N1, -alpha)
56 B[0]=-2*alpha
57 B[N1-1]=0
58
59 #création du vecteur contenant les conditions initiales de la marche
60 U01=np.zeros(N1)
61 U01[0]=1
62 U01[1]=1
63
64 #calcul d'une seule itération pour la diffusion
65 def resol(B, C, A, R ):
66     gam = np.zeros(N1)
67     U=np.zeros(N1)
68     if (B[0]==0):
69         print("ERROR!")
70     bet=B[0]
71     U[0]= (R[0]/(bet))
72     for j in range(1, N1-1):
73         gam[j] = C[j-1]/bet
74         bet=B[j]-A[j]*gam[j]
75         if (bet==0):
76             print("ERROR!")
77         U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
78     k=N1-2
79     while k>=0:
80         U[k] -= gam[k+1]*U[k+1]
81         k = k-1
82     gam=np.zeros(N1)
83     return U
84
85 #boucle pour calculer la diffusion au rang n+1
86 def solutions(MD1, N1, f):
87     if(f==0):
88         return U01
89     U = np.zeros(N1)
90     UI = np.zeros(N1)
91     UI = resol(A, B, C, U01)
92     if (f==1):
93         return UI
94     for i in range(2,nb_iterations+3):
95         U = resol(A, B, C, UI)
96         UI = U
97         if i==(f+1):
98             return U
99
100 solutionsadt= solutions(MD1, N1, nb_iterations)

```

```

101
102 #création des vecteurs vides pour la suite
103 TpreuveTempRT03=[]
104 TpreuveTempn03=[]
105 sommem03 = []
106
107 TpreuveTempRT06=[]
108 TpreuveTempn06=[]
109 sommem06 = []
110
111 TpreuveTempRT15=[]
112 TpreuveTempn15=[]
113 sommem15 = []
114
115 #calcul de la diffusion puis la prolifération avant RT
116 zero = np.zeros(N1)
117 def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
118     oo=np.zeros(N1)
119     for m in range(0, N1-1):
120         oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
121     sommem.extend([oo])
122     TpreuveTempn.extend([oo])
123     TpreuveTempRT.extend([zero])
124     for i in range(1, nbt0):
125         TP= np.zeros(N1)
126         UP=np.zeros(N1)
127         TP = resol(A,B,C, TpreuveTempn[i-1])
128         for j in range(0, N1-1):
129             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
130         sommem.extend([UP])
131         TpreuveTempn.extend([UP])
132         TpreuveTempRT.extend([zero])
133     return TpreuveTempn, TpreuveTempRT, sommem
134 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
135     TpreuveTempRT03, U01, nbt0, sommem03) #remplissage du vecteur
136     diffprolif
137
138 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
139     TpreuveTempRT06, U01, nbt0, sommem06) #remplissage du vecteur
140     diffprolif
141
142 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
143     TpreuveTempRT15, U01, nbt0, sommem15) #remplissage du vecteur
144     diffprolif
145
146
147 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
148     ès RT
149 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
150     oo=np.zeros(N1)
151     op=np.zeros(N1)
152     for i in range(0, N1-1):
153         oo[i]= p*TpreuveTempn[nbt0-1][i]

```

```

145         op[i]= (1-p)*TpreuveTempn[nbt0-1][i]
146
147     Tpreuver = resol(A,B,C, oo)
148     Tpreuven = resol(A,B,C, op)
149     oo = np.zeros(N1)
150     op = np.zeros(N1)
151     s = np.zeros(N1)
152     for m in range(0, N1-1):
153         oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
154         op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
155             Tpreuver[m] )
156         s[m]= op[m] + oo[m]
157
158     sommem.extend([s])
159     TpreuveTempRT.extend([oo])
160     TpreuveTempn.extend([op])
161     for i in range(nbt0+1, nb_iterations):
162
163         TPRT= np.zeros(N1)
164         UPRT=np.zeros(N1)
165         TPRT = resol(A,B,C, TpreuveTempRT[i-1])
166         s = np.zeros(N1)
167         TP= np.zeros(N1)
168         UP=np.zeros(N1)
169         TP = resol(A,B,C, TpreuveTempn[i-1])
170         for j in range(0, N1-1):
171             UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
172             cd= TPRT[j]
173             UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
174             s[j] = UP[j] +UPRT[j]
175
176         sommem.extend([s])
177         TpreuveTempRT.extend([UPRT])
178         TpreuveTempn.extend([UP])
179     return TpreuveTempn, TpreuveTempRT, sommem
180
181 TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M,
182     TpreuveTempRT03, nbt0, sommem03, P03)
183
184 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M,
185     TpreuveTempRT06, nbt0, sommem06, P06)
186
187 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M,
188     TpreuveTempRT15, nbt0, sommem15, P15)
189
190 #Calcul du rayon
191 RXn03= []
192 RYn03 = []
193
194 RXn06= []
195 RYn06 = []

```

```

192 RXn15= []
193 RYn15 = []
194
195 def rayonn(TpreuveTempn, seuill, RXn, RYn):
196     for i in range(0, nb_iterations):
197         r=0
198         for j in range(1, N1-1):
199             if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
200                 >= seuill ) ):
201                 r = (((j-1)+j)/2)*dx
202                 RXn.extend([r])
203                 RYn.extend([i*Delta_t])
204             return RXn, RYn
205
206 RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
207 RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
208 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
209
210 #calcul de la pente associée a la courbe du rayon tumoral
211 def pente(RY, RX):
212     pente = np.zeros(nb_iterations)
213     for i in range(4, nb_iterations-5):
214         pente[i] = (RX[i+4] - RX[i-4]) / (RY[i+4] - RY[i-4])
215     return pente
216
217 pente03 = pente(RYn03, RXn03)
218 max_value03 = max(pente03)
219 maxi03 = (np.where(pente03 == max_value03)[0][0])*Delta_t
220 print("maxi03", maxi03)
221
222 pente06 = pente(RYn06, RXn06)
223 pente06 = pente(RYn06, RXn06)
224 max_value06 = max(pente06)
225 maxi06 = (np.where(pente06 == max_value06)[0][0])*Delta_t
226 print("maxi06", maxi06)
227
228 pente15 = pente(RYn15, RXn15)
229 pente15 = pente(RYn15, RXn15)
230 max_value15 = max(pente15)
231 maxi15 = (np.where(pente15 == max_value15)[0][0])*Delta_t
232 print("maxi15", maxi15)
233
234 #création des figures vides
235 fig4, ax4 = plt.subplots(ncols=1)
236 fig4.subplots_adjust(wspace=0.75)
237
238 #localisation sur les figures
239 left, width = .9, .2
240 bottom, height = .1, 1.5
241 right = left + width
242 top = bottom + height

```

```

242 #représentation graphique
243 ax4.set_title("VARIATION_DE_LA_PENTE")
244 ax4.grid(True)
245 ax4.set_xlim([0, 19])
246 ax4.set_xlabel("Temps_(en_années)")
247 ax4.set_ylabel("Coefficient_de_la_pente")
248 ax4.text(.8, -0.4, "D=1mm^2/an_K=1ans^-1",
249         horizontalalignment='right',
250         verticalalignment='top',
251         transform=ax4.transAxes)
252 ax4.plot(RYn03, pente03, label="Pour_P=0.65_pour_M=2.5")
253 ax4.plot(RYn06, pente06, label="Pour_P=0.85_pour_M=2.5")
254 ax4.plot(RYn15, pente15, label="Pour_P=0.95_pour_M=2.5")
255 ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower_left')
256
257 #affichage des figures
258 fig4.show()

```

2.3.3 Variations des intervalles ΔG , Δg et des valeurs du rayon tumoral

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 #définition des valeurs calculées auparavant
5 RXp = [0.6, 0.7, 0.8, 0.9]
6
7 #pour M=1.5: définition des valeurs calculées auparavant
8 VARDELTAGM15 = [0.48, 0.88, 1.38, 2.14]
9 VARdeltagM15= [0.17, 0.42, 0.65, 1.11]
10 VARrayM15= [6.105, 5.935, 5.585, 4.665 ]
11
12 #pour M=2.5 : définition des valeurs calculées auparavant
13 VARDELTAGM25 = [0.74, 1.07, 1.49, 2.17]
14 VARdeltagM25= [0.31, 0.5, 0.65, 0.97]
15 VARrayM25= [5.885, 5.565, 4.995, 3.525 ]
16
17 #pour M=4.0 : définition des valeurs calculées auparavant
18 VARDELTAGM4 = [0.83, 1.12, 1.51, 2.16]
19 VARdeltagM4= [0.34, 0.45, 0.59, 0.82]
20 VARrayM4=[5.595, 5.155, 4.365, 2.085]
21
22 #création des figures vides
23 fig1, ax1 = plt.subplots()
24 figa, axa = plt.subplots()
25 fig2, ax2 = plt.subplots()
26 figb, axb = plt.subplots()
27 fig3, ax3 = plt.subplots()
28 figc, axc = plt.subplots()
29 fig4, ax4 = plt.subplots()
30 fig4a, ax4a = plt.subplots()
31 fig5, ax5 = plt.subplots()
32
33 #localisation sur les figures
34 left, width = .9, .2
35 bottom, height = .1, 1.5
36 right = left + width
37 top = bottom + height
38
39 #représentation graphique
40 ax1.set_title("Variations de DELTAG et deltag en fonction des valeurs de p
    pour M=1.5 fixé")
41 ax1.set_xlabel("Valeurs de P")
42 ax1.grid(True)
43 ax1.set_ylabel("Intervalle de temps en années")
44 ax1.plot(RXp, VARDELTAGM15, label="Variation de DeltaT")
45 ax1.plot(RXp, VARdeltagM15, label="Variation de deltat")
46 ax1.legend()
47
48 #affichage figures
```



```

49 fig1.show()
50
51 #représentation graphique
52 axa.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de P pour M=1.5 fixé")
53 axa.set_xlabel("Valeurs de P")
54 axa.grid(True)
55 axa.set_ylabel("Rayon minimal (en mm)")
56 axa.plot(RXp, VARrayM15)
57
58 #affichage figures
59 figa.show()
60
61 #représentation graphique
62 ax2.set_title("Variations de DELTAG et deltag en fonction des valeurs de p
    pour M=2.5 fixé")
63 ax2.set_xlabel("Valeurs de P")
64 ax2.grid(True)
65 ax2.set_ylabel("Intervalle de temps en années")
66 ax2.plot(RXp, VARDELTAGM25, label="Variation de DeltaT")
67 ax2.plot(RXp, VARdeltagM25, label="Variation de deltat")
68 ax2.legend()
69
70 #affichage figures
71 fig2.show()
72
73 #représentation graphique
74 axb.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de p pour M=2.5 fixé")
75 axb.set_xlabel("Valeurs de P")
76 axb.grid(True)
77 axb.set_ylabel("Rayon minimal (en mm)")
78 axb.plot(RXp, VARrayM25)
79
80 #affichage des figures
81 figb.show()
82
83 #représentation graphique
84 ax3.set_title("Variations de DELTAG et deltag en fonction des valeurs de p
    pour M=4 fixé")
85 ax3.set_xlabel("Valeurs de P")
86 ax3.set_ylabel("Intervalle de temps en années")
87 ax3.grid(True)
88 ax3.plot(RXp, VARDELTAGM4, label="Variation du DeltaT pour M=4.0 fixé")
89 ax3.plot(RXp, VARdeltagM4, label="Variation du deltat pour M=4.0 fixé")
90 ax3.legend()
91
92 #affichage des figures
93 fig3.show()
94
95 #représentation graphique

```

```

96 axc.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de p pour M=4.0 fixées")
97 axc.set_xlabel("Valeurs de P")
98 axc.grid(True)
99 axc.set_ylabel("Rayon minimal (en mm)")
100 axc.plot(RXp, VARrayM4)
101
102 #affichage figures
103 figc.show()
104
105 #représentation graphique
106 ax4.set_title("Variations de DELTAG fonction des valeurs de p pour plusieurs
    valeurs de M fixées")
107 ax4.set_xlabel("Valeurs de P")
108 ax4.set_ylabel("Intervalle de temps en années")
109 ax4.grid(True)
110 ax4.plot(RXp, VARDELTAGM15, label="Variation du Delta T pour M=1.5 fixé")
111 ax4.plot(RXp, VARDELTAGM25, label="Variation du Delta T pour M=2.5 fixé")
112 ax4.plot(RXp, VARDELTAGM4, label="Variation du Delta T pour M=4.0 fixé")
113
114 #affichage figures
115 fig4.show()
116
117 #représentation graphique
118 ax4a.set_title("Variations de deltag en fonction des valeurs de p pour
    plusieurs valeurs de M fixées")
119 ax4a.set_xlabel("Valeurs de p")
120 ax4a.set_ylabel("Intervalle de temps en années")
121 ax4a.grid(True)
122 ax4a.plot(RXp, VARdeltagM15, label="Variation du deltat pour M=1.5 fixé")
123 ax4a.plot(RXp, VARdeltagM25, label="Variation du deltat pour M=2.5 fixé")
124 ax4a.plot(RXp, VARdeltagM4, label="Variation du deltat pour M=4.0 fixé")
125 ax4a.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
126
127 #affichages figures
128 fig4a.show()
129
130
131 #représentation graphique
132 ax5.set_title("Variations de la valeur du rayon minimal en fonction des
    valeurs de P pour plusieurs valeurs de M fixées" )
133 ax5.set_xlabel("Valeurs de P")
134 ax5.grid(True)
135 ax5.set_ylabel("Rayon minimal (en mm)")
136 ax5.plot(RXp, VARrayM15, label="Variation du rayon pour M=1.5 fixé")
137 ax5.plot(RXp, VARrayM25, label="Variation du rayon pour M=2.5 fixé")
138 ax5.plot(RXp, VARrayM4, label="Variation du rayon pour M=4 fixé")
139 ax5.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
140
141 #affichage des figures
142 fig5.show()

```