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

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 :

```
import numpy as np
import matplotlib.pyplot as plt

#définition des paramètres du temps
Tempstot = 40 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
##de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))


#parameèer L= taille physique boite(mm)/
L= 45 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2 #en mm**2
N1 = int(L/dx) #taille de la representation de l'espace
print("N1= ", N1)
X = np.arange(0, L, dx)
print("X= ", X)

D = 1 #coeff de diffusion mm^2/ans
alpha = (D*(Delta_t)/dx2) #defini par rapport àl'intervalle de temps, dx et
    D
k=1 #coeff de prolifération (en ans^-1)
print("alpha=", alpha)

#création des figures vides
fig, ax1 = plt.subplots(ncols=1)
fig.subplots_adjust(wspace=0.75)
fig2, ax2 = plt.subplots(ncols=1)
fig2.subplots_adjust(wspace=0.75)
fig3, ax3 = plt.subplots(ncols=1)
fig3.subplots_adjust(wspace=0.75)

#matrice M1D pour les prochains calculs
def MatriceM1(N1, alpha):
    MD1 = np.zeros((N1,N1))

    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= 0. -2*alpha
    MiD1= np.linalg.inv(MD1)
```

```
    return MD1 , MiD1

MD1, MiD1 = MatriceM1(N1, alpha)
print("MD1=␣", MD1)

#Matrice avec valeurs initiales de U0 àpartir
#des conditions initiales données par la gaussienne
def MatriceU0(N1, X, t0):
    U0=np.zeros(N1)
    for i in range (0, N1-1):
       U0[i] = np.exp(-(X[i]**2)/(4*D*t0))
    return U0

U0 = MatriceU0(N1, X, 0.1)

#définition de la solution exacte de la diffusion pour comparer

def sol_exacteD(X, t0, t):
    sol_exacteD=np.zeros(N1)
    for i in range(0, N1-1):
        sol_exacteD[i] =
        ((t0/(t-t0))**(1/2))*(np.exp( -(X[i]**2) / (4*D*(t+t0))))
    return sol_exacteD
sol_exacteD = sol_exacteD(X, 0.1, Tempstot)

#création d'un vecteur vide que l'on remplira par la suite
Tpreuve = []

def ResD( Tpreuve):
    UT=np.zeros(N1)
    Tpreuve.extend([U0])
    UT=U0
    U=np.zeros(N1)
    for i in range(0, nb_iterations-1):
        U= np.dot(MiD1, UT)
        UT = U
        Tpreuve.extend([U])
    return Tpreuve
Tpreuve = ResD(Tpreuve)
print(np.shape(Tpreuve))


#définition de la solution exacte de la diffusion+prolif pour comparer
def sol_exacteT(X, t0, t):
    sol_exacteT=np.zeros(N1)
    for i in range(0, N1-1):
        sol_exacteT[i] = ((t0/(t+t0))**(1/2)) * (np.exp( -(X[i]**2) / (4*D*(
            t+t0)))) * (np.exp(k*t) )
    return sol_exacteT

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␣mm)")
133 ax1.set_xlim([0,10])
134 ax1.text(1.1, -0.3, "D=1.0␣mm^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=2*dt")
141 ax1.plot(X, Tpreuve[200], label="␣Sol_approch␣diff␣t=20*dt")
142 ax1.plot(X, Tpreuve[500], label="␣Sol_approch␣diff␣t=50*dt")
143 ax1.plot(X, Tpreuve[1000], label="␣Sol_approch␣diff␣t=100*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␣ans^-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 :

```
import numpy as np
import matplotlib.pyplot as plt

#paramètres généraux du temps
Tempstot = 30 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance
##en effet de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

#paramètres généraux de l'espace et autres
#calculer L= taille physique boite(mm)/
L= 70 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la représentation
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion (en mm^2/ans)
k=1 #coeff de prolifération (en ans^-1)
alpha = (D*Delta_t)/dx2 #défini par rapport àl'intervalle de temps, dx et D
seuil = np.full(N1, 0.07)
seuill = 0.07 #valeur du seuil de detection
vitfront = 2*np.sqrt(D*k) #vitesse de front 2sqrt(Dk)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k)) #vitesse de front + 10%
vit_front=np.full(nb_iterations-1, 2*np.sqrt(D*k)) #fonction constante avec
    la vitesse de front
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%

#matrice M1D pour la modélisation
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
    return MD1

MD1 = MatriceMD1(N1, alpha)
print("MD1= ", MD1)

#on stocke les diagonales dans les nouveaux vecteurs vides
A=np.full(N1, 1+2*alpha)
C=np.full(N1, -alpha)
C[0]=0
```

```
B=np.full(N1, -alpha)
B[0]=-2*alpha
B[N1-1]=0

#Matrice avec conditions initiales U0 données par la marche
U01=np.zeros(N1)
U01[0]=1
U01[1]=1

#calcul de la diffusion une seule itération
def resol(B, C, A, R ):
    gam = np.zeros(N1)
    U=np.zeros(N1)
    if (B[0]==0):
        print("ERROR!")
    bet=B[0]
    U[0]= (R[0]/(bet))
    for j in range(1, N1-1):
        gam[j] = C[j-1]/bet
        bet=B[j]-A[j]*gam[j]
        if (bet==0):
            print("ERROR!")
        U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
    k=N1-2
    while k>=0:
        U[k] -= gam[k+1]*U[k+1]
        k = k-1
    gam=np.zeros(N1)
    return U

#boucle pour calculer la diffusion au rang n+1
def solutions de la diffusion (MD1, N1, f):
    if(f==0):
        return U01
    U = np.zeros(N1)
    UI = np.zeros(N1)
    UI = resol(A, B, C, U01)
    if (f==1):
        return UI
    for i in range(2,nb_iterations+3):
         U = resol(A, B, C, UI)
         UI = U
         if i==(f):
             return U

solutionsadt= solutions(MD1, N1, nb_iterations-1)

#def d'un tableau vide que l'on remplira avec les valeurs de la diffprolif a
    tout temps fixé
TpreuveTemp=[]
#fonction calculant les valeurs de la diffusion puis de la proliferation
```

```
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 = []
```

```
    g=0
    vit0=[]
    for i in range(0, nb_iterations-1):
        if ( ( pente[i] <= vitfront ) and ( pente[i-1] >= vitfront ) ):
            g= (((i-1)+i)/2)*Delta_t
            vit0.extend([g])

        elif(( pente[i] <= vitfrontdix ) and ( pente[i-1] >= vitfrontdix )):
            e= (((i-1)+i)/2)*Delta_t
            vit10.extend([e])

    return vit0, vit10
g, e = vitfrontpente(pente, vitfront)

#création des figures vides
fig1, ax1 = plt.subplots(ncols=1)
fig1.subplots_adjust(wspace=0.75)
fig2, ax2 = plt.subplots(ncols=1)
fig2.subplots_adjust(wspace=0.75)
fig3, ax3 = plt.subplots(ncols=1)
fig3.subplots_adjust(wspace=0.75)

#définition des localisations sur les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height

#représentation graphique
ax1.set_title("DIFFUSION␣PROLIFERATION␣SANS␣RT")
ax1.set_xlabel("Longueur␣de␣la␣boîte␣(en␣mm)")
ax1.set_ylabel("Densité␣cellulaire")
ax1.text(.8, -0.4, "D=1mm^2/an␣K=1␣ans^-1",
       horizontalalignment='right',
       verticalalignment='top',
       transform=ax1.transAxes)
ax1.plot(X, TpreuveTemp[0], label="Sol_approch␣t=0dt")
ax1.plot(X, TpreuveTemp[20], label="Sol_approch␣t=20dt")
ax1.plot(X, TpreuveTemp[75], label="Sol_approch␣t=75dt")
ax1.plot(X, TpreuveTemp[250], label="Sol_approch␣t=250dt")
ax1.plot(X, TpreuveTemp[750], label="Sol_approch␣t=750dt")
ax1.plot(X, TpreuveTemp[1500], label="Sol_approch␣t=1500dt")
ax1.plot(X, TpreuveTemp[2500], label="Sol_approch␣t=2500dt")
ax1.plot(X, TpreuveTemp[nb_iterations-2], label="Sol_approch␣t=Tempstot")
ax1.plot(X, seuil, label="Seuil=0.07mm")
ax1.legend(bbox_to_anchor=(-0.7, -0.7, -0.7, -0.7), loc='lower␣left')

ax2.set_title("VARIATION␣DU␣RAYON␣TUMORAL")
ax2.set_xlabel("Temps␣(en␣années)")
ax2.set_ylabel("Rayon␣de␣la␣tumeur␣(en␣mm)")
ax2.text(.8, -0.4, "D=1mm^2/an␣K=1␣ans^-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=1 ans^-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
    =(6,7.4), arrowprops={'facecolor':'black', 'shrink':0.05} )
215 ax3.plot(RY, pente, label="Variation de la pente en fonction du temps")
216 ax3.plot(RY, vit_front, label=r'$ f(x)= 2\sqrt{Dk}$')
217 ax3.plot(RY, vit_frontdix, label=r' g(x)= $2\sqrt{Dk}$ + 10%')
218 ax3.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
219 
220 #affichage figures
221 
222 fig1.show()
223 fig2.show()
224 fig3.show()
```

2 Application de la Radiothérapie :

2.1 Processus de diffusion proliferation suite à une session de RT

```
import numpy as np
import matplotlib.pyplot as plt
\\
Tempstot = 30 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
#de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height


#paramètres pour l'espace et autres
#calculer L= taille physique boîte (mm)/
L= 70 en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la representation de l'espace
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion
k=1 #coeff de prolifération
P15=0.9 #proportion des cellules atteintes par la RT
M=2.5 #taux de mort des cellules atteintes par la RT (en ans^-1)
nbt0 = 950 #Moment de l'appli de la RT
alpha = (D*Delta_t)/dx2 #defini par rapport àl'intervalle de temps et dx
print(alpha)
seuil = np.full(N1, 0.07)
seuill = 0.07
vitfront = 2*np.sqrt(D*k)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))


#matrice M1D pour les prochains calculs
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
```

```
    return MD1

MD1 = MatriceMD1(N1, alpha)
print("MD1= ", MD1)

#on stocke les diagonales dans les nouveaux vecteurs vides
A=np.full(N1, 1+2*alpha)
C=np.full(N1, -alpha)
C[0]=0
B=np.full(N1, -alpha)
B[0]=-2*alpha
B[N1-1]=0

#vecteur avec les conditions initiales données par la marche
U01=np.zeros(N1)
U01[0]=1
U01[1]=1

#fonction diffusion une seule itération
def resol(B, C, A, R ):
    gam = np.zeros(N1)
    U=np.zeros(N1)
    if (B[0]==0):
        print("ERROR!")
    bet=B[0]
    U[0]= (R[0]/(bet))
    for j in range(1, N1-1):
        gam[j] = C[j-1]/bet
        bet=B[j]-A[j]*gam[j]
        if (bet==0):
            print("ERROR!")
        U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
    k=N1-2
    while k>=0:
        U[k] -= gam[k+1]*U[k+1]
        k = k-1
    gam=np.zeros(N1)
    return U

#boucle pour calculer au rang n+1 les solutions de la diffusion
def solutions(MD1, N1, f):
    if(f==0):
        return U01
    U = np.zeros(N1)
    UI = np.zeros(N1)
    UI = resol(A, B, C, U01)
    if (f==1):
        return UI
    for i in range(2,nb_iterations+3):
         U = resol(A, B, C, UI)
         UI = U
```

```
        if i==(f+1):
            return U

solutionsadt= solutions(MD1, N1, nb_iterations)

#fonction de la diffusion prolifération avant RT
TpreuveTempRT15=[]
TpreuveTempn15=[]
sommem15 = []

zero = np.zeros(N1)
def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
    oo=np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
    sommem.extend([oo])
    TpreuveTempn.extend([oo])
    TpreuveTempRT.extend([zero])
    for i in range(1, nbt0):
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
        sommem.extend([UP])
        TpreuveTempn.extend([UP])
        TpreuveTempRT.extend([zero])
    return TpreuveTempn, TpreuveTempRT, sommem

TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
    TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
    diffprolif


#fonction calculant les valeurs de la diffusion puis de la prolifération apr
    ès RT
def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
    oo=np.zeros(N1)
    op=np.zeros(N1)
    for i in range(0, N1-1):
        oo[i]= p*TpreuveTempn[nbt0-1][i]
        op[i]= (1-p)*TpreuveTempn[nbt0-1][i]

    Tpreuver = resol(A,B,C, oo)
    Tpreuven = resol(A,B,C, op)
    oo = np.zeros(N1)
    op = np.zeros(N1)
    s = np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
```

```
        op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
        s[m]= op[m] + oo[m]

    sommem.extend([s])
    TpreuveTempRT.extend([oo])
    TpreuveTempn.extend([op])
    for i in range(nbt0+1, nb_iterations):

        TPRT= np.zeros(N1)
        UPRT=np.zeros(N1)
        TPRT = resol(A,B,C, TpreuveTempRT[i-1])
        s = np.zeros(N1)
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
            cd= TPRT[j]
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
            s[j] = UP[j] +UPRT[j]

        sommem.extend([s])
        TpreuveTempRT.extend([UPRT])
        TpreuveTempn.extend([UP])
    return TpreuveTempn, TpreuveTempRT, sommem


TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M,
    TpreuveTempRT15, nbt0, sommem15, P15)


#Calcul du rayon

RXn15= []
RYn15 = []

def rayonn(TpreuveTempn, seuill, RXn, RYn):
    for i in range(0, nb_iterations):
        r=0
        for j in range(1, N1-1):
            if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
                >= seuill ) ):
                r = (((j-1)+j)/2)*dx
        RXn.extend([r])
        RYn.extend([i*Delta_t])
    return RXn, RYn

RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)

```

```
#Calcul de la valeur minimale du rayon et deltag (intervalle de temps pour atteindre cette dernière)
def minn(RX):
    minim=35
    indice = 0
    deltag=0
    for i in range (nbt0, nb_iterations):
        m = RX[i]
        if (m<minim):
            minim = m
            indice = i *Delta_t
            deltag= (i*Delta_t) - (nbt0*Delta_t)
    return minim, indice, deltag


minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la valeur minimum du rayon ainsi que le moment m auquel cette valeur min est atteinte
print("minimRX", minimRX15)
print("indice", indicemin15)
print("deltag", deltag15)

#calcul de l'intervalle de temps DeltaG
def momentegal(RX):
    bingo=RX[nbt0]
    DELTAG=0
    for i in range(nbt0, nb_iterations):
        if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
            DELTAG = (i*Delta_t) - (nbt0*Delta_t)
    return DELTAG


DELTAG15 = momentegal(RXn15)
print("DELTAG15 = ",DELTAG15)

#création des figures vides
fig2, ax2 = plt.subplots(ncols=1)
fig2.subplots_adjust(wspace=0.75)

fig4, ax4 = plt.subplots(ncols=1)
fig4.subplots_adjust(wspace=0.75)

#représentation graphique

ax2.set_title(" DIFFUSION PROLIFERATION CELLULES ")
ax2.grid(True)
ax2.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 P=0.9 M=2.5 ans^-1",
        horizontalalignment='right',
        verticalalignment='top',
        transform=ax2.transAxes)
ax2.set_xlabel("Longueur de la boîte (en mm)")
```

```
ax2.set_ylabel("Densité cellulaire")
ax2.plot(X, sommem15[0], label="sol_approch c t=0dt", color='gray' )
ax2.plot(X, sommem15[20], label="sol_approch c t=20dt", color='gray')
ax2.plot(X, sommem15[50], label="sol_approch c t=50dt", color='gray')
ax2.plot(X, sommem15[250], label="sol_approch c t=250dt", color='gray')
ax2.plot(X, sommem15[400], label="sol_approch ct=400dt ", color='gray')
ax2.plot(X, sommem15[550], label="sol_approch ct=550dt", color='gray')
ax2.plot(X, sommem15[625], label="sol_approch ct=625dt", color='gray')
ax2.plot(X, sommem15[750], label="sol_approch c  t=750dt", color='gray')
ax2.plot(X, sommem15[875], label="sol_approch c  t=875dt", color='gray')
ax2.plot(X, sommem15[950], label="sol_approch c moment irradiation RT t=950
    dt", color='red')
ax2.plot(X, sommem15[1000], label="sol_approch c  t=1000dt", color='orange')
ax2.plot(X, sommem15[1150], label="sol_approch c  t=1150dt", color='pink')
ax2.plot(X, sommem15[1300], label="sol_approch c t=1300dt ", color='magenta'
    )
ax2.plot(X, sommem15[1500], label="sol_approch c t=1500dt ", color='purple')
ax2.plot(X, sommem15[1750], label="sol_approch c t=1750dt ", color='blue')
ax2.plot(X, sommem15[2000], label="sol_approch c t=2000dt ", color='green')
ax2.plot(X, sommem15[2250], label="sol_approch c t=2250dt ", color='black')
ax2.plot(X, sommem15[2250], label="sol_approch c t=2750dt ", color='black')
ax2.plot(X, sommem15[nb_iterations-1], label="sol_approch c t=Tempstot",
    color='black')
ax2.legend(bbox_to_anchor=(-0.99, -0.99, -0.99, -0.99), loc='lower left')


ax4.set_title("VARIATION DU RAYON TUMORAL ")
ax4.grid(True)
ax4.set_xlabel("Temps (en années)")
ax4.set_ylabel("Rayon de la tumeur (en mm)")
ax4.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 M=1.1 ans^-1 fixé",
        horizontalalignment='right',
        verticalalignment='top',
        transform=ax4.transAxes)
ax4.annotate('Session de radiothérapie', xy=(9,7.1), xytext=(2.5,17.2),
    arrowprops={'facecolor':'black', 'shrink':0.05} )
ax4.plot(RYn15, RXn15, label="P=0.9")
ax4.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')

#affichage figures
fig2.show()
fig4.show()
```

2.2 Étude du paramètre M

2.2.1 Variation du Rayon tumoral pour plusieurs valeurs de M

```
import numpy as np
import matplotlib.pyplot as plt

#définition des paramètres temps
Tempstot = 20 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
#de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

#paramètres de l'espace et autres
#calculer L= taille physique boite(mm)/
L= 70 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la représentation de l'espace
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion
k=1 #coeff de prolifération
M03=1.5 #taux de mort
M06=2.5 #taux de mort
M15=4 #taux de mort
p=0.95 #proportion de cellules atteintes par la RT
 #proportion des cellules atteintes par la RT
nbt0 = 950 #moment de l'appli de la RT
alpha = (D*Delta_t)/dx2 #defini par rapport àl'intervalle de temps, dx et D
print(alpha)
seuil = np.full(N1, 0.07)
seuill = 0.07 #seuil de détection
vitfront = 2*np.sqrt(D*k)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))


#matrice M1D pour les prochains calculs
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
    return MD1

MD1 = MatriceMD1(N1, alpha)
```

```
print("MD1= ", MD1)

#on stocke les diagonales dans les nouveaux vecteurs vides
A=np.full(N1, 1+2*alpha)
C=np.full(N1, -alpha)
C[0]=0
B=np.full(N1, -alpha)
B[0]=-2*alpha
B[N1-1]=0

#conditions initiales données par la marche
U01=np.zeros(N1)
U01[0]=1
U01[1]=1

#fonction diffusion pour une seule itération
def resol(B, C, A, R ):
    gam = np.zeros(N1)
    U=np.zeros(N1)
    if (B[0]==0):
        print("ERROR!")
    bet=B[0]
    U[0]= (R[0]/(bet))
    for j in range(1, N1-1):
        gam[j] = C[j-1]/bet
        bet=B[j]-A[j]*gam[j]
        if (bet==0):
            print("ERROR!")
        U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
    k=N1-2
    while k>=0:
        U[k] -= gam[k+1]*U[k+1]
        k = k-1
    gam=np.zeros(N1)
    return U

#boucle pour calculer la diffusion au rang n+1
def solutions(MD1, N1, f):
    if(f==0):
        return U01
    U = np.zeros(N1)
    UI = np.zeros(N1)
    UI = resol(A, B, C, U01)
    if (f==1):
        return UI
    for i in range(2,nb_iterations+3):
         U = resol(A, B, C, UI)
         UI = U
         if i==(f+1):
             return U

```

```
solutionsadt= solutions(MD1, N1, nb_iterations)

#création des vecteurs vides que l'on utilisera par la suite:
TpreuveTempRT03=[]
TpreuveTempn03=[]
sommem03 = []

TpreuveTempRT06=[]
TpreuveTempn06=[]
sommem06 = []

TpreuveTempRT15=[]
TpreuveTempn15=[]
sommem15 = []

zero = np.zeros(N1)
#modélisation de la diffusion prolifération avant RT
def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
    oo=np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
    sommem.extend([oo])
    TpreuveTempn.extend([oo])
    TpreuveTempRT.extend([zero])
    for i in range(1, nbt0):
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
        sommem.extend([UP])
        TpreuveTempn.extend([UP])
        TpreuveTempRT.extend([zero])
    return TpreuveTempn, TpreuveTempRT, sommem
TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
    TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
    diffprolif
TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
    TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
    diffprolif
TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
    TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
    diffprolif


#fonction calculant les valeurs de la diffusion puis de la prolifération apr
    ès RT

def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem):
    oo=np.zeros(N1)
    op=np.zeros(N1)
```

```
    for i in range(0, N1-1):
        oo[i]= p*TpreuveTempn[nbt0-1][i]
        op[i]= (1-p)*TpreuveTempn[nbt0-1][i]

    Tpreuver = resol(A,B,C, oo)
    Tpreuven = resol(A,B,C, op)
    oo = np.zeros(N1)
    op = np.zeros(N1)
    s = np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
        op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
        s[m]= op[m] + oo[m]

    sommem.extend([s])
    TpreuveTempRT.extend([oo])
    TpreuveTempn.extend([op])
    for i in range(nbt0+1, nb_iterations):

        TPRT= np.zeros(N1)
        UPRT=np.zeros(N1)
        TPRT = resol(A,B,C, TpreuveTempRT[i-1])
        s = np.zeros(N1)
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
            cd= TPRT[j]
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
            s[j] = UP[j] +UPRT[j]

        sommem.extend([s])
        TpreuveTempRT.extend([UPRT])
        TpreuveTempn.extend([UP])
    return TpreuveTempn, TpreuveTempRT, sommem

TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M03
    , TpreuveTempRT03, nbt0, sommem03)
TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M06
    , TpreuveTempRT06, nbt0, sommem06)
TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M15
    , TpreuveTempRT15, nbt0, sommem15)


#Calcul du rayon
RXn03= []
RYn03 = []

RXn06= []
```

```
RYn06 = []

RXn15= []
RYn15 = []

def rayonn(TpreuveTempn, seuill, RXn, RYn):
    for i in range(0, nb_iterations):
        r=0
        for j in range(1, N1-1):
            if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
                >= seuill ) ):
                r = (((j-1)+j)/2)*dx
        RXn.extend([r])
        RYn.extend([i*Delta_t])
    return RXn, RYn

RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)

#On cherche a trouver le moment où le rayon est minimal suite a la session
    de RT (deltag) et la valeur minimale du rayon
def minn(RX):
    minim=35
    indice = 0
    deltag=0
    for i in range (nbt0, nb_iterations):
        m = RX[i]
        if (m<minim):
            minim = m
            indice = i *Delta_t
            deltag= (i*Delta_t) - (nbt0*Delta_t)
    return minim, indice, deltag

minimRX03, indicemin03, deltag03 = minn(RXn03) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
print("minimRX", minimRX03)
print("indice", indicemin03)
print("deltag", deltag03)


minimRX06, indicemin06, deltag06 = minn(RXn06) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
print("minimRX", minimRX06)
print("indice", indicemin06)
print("deltag", deltag06)

minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
```

```
    est atteinte
print("minimRX", minimRX15)
print("indice", indicemin15)
print("deltag", deltag15)

#calcul de l'intervalle DeltaG
def momentegal(RX):
    bingo=RX[nbt0]
    DELTAG=0
    for i in range(nbt0, nb_iterations):
        if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
            DELTAG = (i*Delta_t) - (nbt0*Delta_t)
    return DELTAG

DELTAG03 = momentegal(RXn03)
print("DELTAG03 = ",DELTAG03)

DELTAG06 = momentegal(RXn06)
print("DELTAG06 = ",DELTAG06)

DELTAG15 = momentegal(RXn15)
print("DELTAG15 = ",DELTAG15)

#création des figures vides
fig4, ax4 = plt.subplots(ncols=1)
fig4.subplots_adjust(wspace=0.75)

#localisation dans les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height

#représentation graphique
ax4.set_title("VARIATION DU RAYON TUMORAL ")
ax4.grid(True)
ax4.set_xlim([0, 18])
ax4.set_ylim([0, 25])
ax4.set_xlabel("Temps (en années)")
ax4.set_ylabel("Rayon de la tumeur (en mm)")
ax4.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 P=0.95 fixés RT=9.5 ans",
        horizontalalignment='right',
        verticalalignment='top',
        transform=ax4.transAxes)
ax4.annotate('Session de radiothérapie', xy=(9.5,15.1), xytext=(3.5,20.2),
    arrowprops={'facecolor':'black', 'shrink':0.05} )
ax4.plot(RYn03, RXn03, label="M=1.1")
ax4.plot(RYn06, RXn06, label="M=2.5")
ax4.plot(RYn15, RXn15, label="M=4")
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

```
import numpy as np
import matplotlib.pyplot as plt


#paramètres du temps
Tempstot = 20 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance
#en effet de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

#paramètres de l'espace et autres
#calculer L= taille physique boîte(mm)/
L= 70 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la représentation de l'espace
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion
k=1 #coeff de prolifération
P=0.95 #proportion de cellules atteintes par la RT
M15=1.1 #taux de mort
M25=2.5 #taux de mort
M4=4 #taux de mort
nbt0 = 950 #appli de la RT
alpha = (D*Delta_t)/dx2 #défini par rapport àl'intervalle de temps, dx et D
print(alpha)
seuil = np.full(N1, 0.07)
seuill = 0.07 #seuil de détection
vitfront = 2*np.sqrt(D*k)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))


#matrice M1D pour les prochains calculs
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
    return MD1

MD1 = MatriceMD1(N1, alpha)
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,
    TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
    diffprolif
135 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
    TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
    diffprolif
136 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
    TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
    diffprolif
137
138
139 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
    ès RT
140 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
141     oo=np.zeros(N1)
142     op=np.zeros(N1)
143     for i in range(0, N1-1):
144         oo[i]= p*TpreuveTempn[nbt0-1][i]
```

```
        op[i]= (1-p)*TpreuveTempn[nbt0-1][i]

    Tpreuver = resol(A,B,C, oo)
    Tpreuven = resol(A,B,C, op)
    oo = np.zeros(N1)
    op = np.zeros(N1)
    s = np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
        op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
        s[m]= op[m] + oo[m]

    sommem.extend([s])
    TpreuveTempRT.extend([oo])
    TpreuveTempn.extend([op])
    for i in range(nbt0+1, nb_iterations):

        TPRT= np.zeros(N1)
        UPRT=np.zeros(N1)
        TPRT = resol(A,B,C, TpreuveTempRT[i-1])
        s = np.zeros(N1)
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
            cd= TPRT[j]
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
            s[j] = UP[j] +UPRT[j]

        sommem.extend([s])
        TpreuveTempRT.extend([UPRT])
        TpreuveTempn.extend([UP])
    return TpreuveTempn, TpreuveTempRT, sommem

TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M15
    , TpreuveTempRT03, nbt0, sommem03, P)
TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M25
    , TpreuveTempRT06, nbt0, sommem06, P)
TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M4,
     TpreuveTempRT15, nbt0, sommem15, P)


#Calcul du rayon
RXn03= []
RYn03 = []

RXn06= []
RYn06 = []

```

```
RXn15= []
RYn15 = []

def rayonn(TpreuveTempn, seuill, RXn, RYn):
    for i in range(0, nb_iterations):
        r=0
        for j in range(1, N1-1):
            if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
                >= seuill ) ):
                r = (((j-1)+j)/2)*dx
        RXn.extend([r])
        RYn.extend([i*Delta_t])
    return RXn, RYn

RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)

#calcul de la pente associée àla courbe du rayon
def pente(RY, RX):
    pente = np.zeros(nb_iterations)
    for i in range(4, nb_iterations-5):
        pente[i] = (RX[i+4] - RX[i-4]) / (RY[i+4] - RY[i-4])
    return pente
pente03 = pente(RYn03, RXn03)
max_value03 = max(pente03)
maxi03 = (np.where(pente03 == max_value03)[0][0])*Delta_t
print("maxi03", maxi03)

pente06 = pente(RYn06, RXn06)
pente06 = pente(RYn06, RXn06)
max_value06 = max(pente06)
maxi06 = (np.where(pente06 == max_value06)[0][0])*Delta_t
print("maxi06", maxi06)

pente15 = pente(RYn15, RXn15)
pente15 = pente(RYn15, RXn15)
max_value15 = max(pente15)
maxi15 = (np.where(pente15 == max_value15)[0][0])*Delta_t
print("maxi15", maxi15)

#création des figures vides
fig4, ax4 = plt.subplots(ncols=1)
fig4.subplots_adjust(wspace=0.75)

#localisations sur les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height

```

```
#représentation graphique
ax4.set_title("VARIATION DE LA PENTE")
ax4.grid(True)
ax4.set_xlabel("Temps (en années)")
ax4.set_xlim([0, 19])
ax4.set_ylabel("Coefficient de la pente")
ax4.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 fixées",
       horizontalalignment='right',
       verticalalignment='top',
       transform=ax4.transAxes)
ax4.plot(RYn03, pente03, label="Pour M=1.1 pour P=0.95")
ax4.plot(RYn06, pente06, label="Pour M=2.5 pour P=0.95")
ax4.plot(RYn15, pente15, label="Pour M=4 pour P=0.95")
ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower left')

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

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

```
import numpy as np
import matplotlib.pyplot as plt

#valeurs qu'on a calculé auparavant
RXmp06 = [0.6, 0.7, 0.8, 0.9, 0.99, 1.5, 2.5, 3.0, 4.0]
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]

#pour P=0.6
VARRAYMINP06 = [6.165, 6.165,6.165, 6.165, 6.165, 6.105, 5.885, 5.775,
    5.595]
VARDELTATP06 = [0.0005,0.001,0.05, 0.07, 0.1, 0.48, 0.74,0.79, 0.83]
VARdeltatP06= [0, 0, 0,0, 0, 0.17, 0.31, 0.35, 0.34]

#pour P=0.7
VARRAYMINP07 = [6.165, 6.165,6.165, 6.145, 6.115, 5.935, 5.565, 5.415,
    5.155]
VARDELTATP07 = [0.0005,0.03,0.2, 0.42, 0.53, 0.88, 1.07,1.1, 1.12]
VARdeltatP07= [0, 0, 0,0.07, 0.23, 0.42, 0.5, 0.46, 0.45]


#pour P=0.8
VARRAYMINP08 = [6.145, 6.105,6.055, 5.995, 5.935, 5.585, 4.995, 4.755,
    4.365]
VARDELTATP08 = [0.43,0.71,0.9, 1.05, 1.14, 1.38, 1.49,1.5, 1.51]
VARdeltatP08= [0.06, 0.31, 0.4,0.47, 0.53, 0.65, 0.65, 0.64, 0.59]

#pour P=0.9
VARRAYMINP09seul = [ 6.155,6.155 ,6.135 , 6.045,5.925, 5.785,5.645, 5.495,
    5.365, 4.665, 3.525, 3.025, 2.085]
VARDELTATP09seul = [0.05,0.1,0.75 , 1.32, 1.6,1.77,1.9, 1.97, 2.02, 2.14,
    2.17,2.17, 2.16]
VARdeltatP09seul= [0, 0, 0.28,0.73 , 0.86, 1.02, 1.02, 1.11, 1.11, 1.11,
    0.97, 0.92, 0.82]

#valeurs calculées auparavant
VARRAYMINP09 = [5.925, 5.785,5.645, 5.495, 5.365, 4.665, 3.525, 3.025,
    2.085]
VARDELTATP09 = [1.6,1.77,1.9, 1.97, 2.02, 2.14, 2.17,2.17, 2.16]
VARdeltatP09= [0.86, 1.02, 1.02, 1.11, 1.11, 1.11, 0.97, 0.92, 0.82]

#création des figures vides
fig1, ax1 = plt.subplots()
figa, axa = plt.subplots()
fig2, ax2 = plt.subplots()
figb, axb = plt.subplots()
fig3, ax3 = plt.subplots()
figc, axc = plt.subplots()
fig4, ax4 = plt.subplots()
figd, axd = plt.subplots()
```

```
fig5, ax5 = plt.subplots()
fig6, ax6 = plt.subplots()
fig7, ax7 = plt.subplots()

#représentation graphique
ax1.set_title("Variations de DELTAT et deltat en fonction des valeurs de M pour P=0.6 fixé")
ax1.set_xlabel("Valeurs de M")
ax1.grid(True)
ax1.set_ylabel("Intervalle de temps en années")
ax1.plot(RXmp06, VARDELTATP06, label="Variation de DeltaT")
ax1.plot(RXmp06, VARdeltatP06, label="Variation de deltat")
ax1.legend()

#affichage des figures
fig1.show()

#représentation graphique
axa.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de M pour P=0.6 fixé")
axa.set_xlabel("Valeurs de M")
axa.grid(True)
axa.set_ylabel("Rayon minimal (en mm)")
axa.plot(RXmp06, VARRAYMINP06)

#affichage figures
figa.show()

#représentation graphique
ax2.set_title("Variations de DELTAT et deltat en fonction des valeurs de M pour P=0.7 fixé")
ax2.set_xlabel("Valeurs de M")
ax2.grid(True)
ax2.set_ylabel("Intervalle de temps en années")
ax2.plot(RXmp06, VARDELTATP07, label="Variation de DeltaT")
ax2.plot(RXmp06, VARdeltatP07, label="Variation de deltat")
ax2.legend()

#affichage des figures
fig2.show()

#représentation graphique
axb.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de M pour P=0.7 fixé")
axb.set_xlabel("Valeurs de M")
axb.grid(True)
axb.set_ylabel("Rayon minimal (en mm)")
axb.plot(RXmp06, VARRAYMINP07)

#affichage des figures
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, VARDELTATP08, 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, VARDELTATP09seul, 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_iches='tight')
134
135 #affichage des figures
```

```
ax5.set_title("Variations de DELTAT en fonction des valeurs de M pour plusieurs valeurs de p fixées")
ax5.set_xlabel("Valeurs de M")
ax5.set_ylabel("Intervalle de temps en années")
ax5.grid(True)
ax5.plot(RXmp06, VARDELTATP06, label="Variation de DeltaT pour P=0.6fixé")
ax5.plot(RXmp06, VARDELTATP07, label="Variation de DeltaT pour P=0.7 fixé")
ax5.plot(RXmp06, VARDELTATP08, label="Variation de DeltaT pour P=0.8 fixé")
ax5.plot(RXmp06, VARDELTATP09, label="Variation de DeltaT pour P=0.9 fixé")
ax5.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')
fig5.show()

#représentation graphique
ax6.set_title("Variations de deltat en fonction des valeurs de M pour plusieurs valeurs de p fixées")
ax6.set_xlabel("Valeurs de M")
ax6.set_ylabel("Intervalle de temps en années")
ax6.grid(True)
ax6.plot(RXmp06, VARdeltatP06, label="Variation de deltat pour P=0.6 fixé")
ax6.plot(RXmp06, VARdeltatP07, label="Variation de deltat pour P=0.7 fixé")
ax6.plot(RXmp06, VARdeltatP08, label="Variation de deltat pour P=0.8 fixé")
ax6.plot(RXmp06, VARdeltatP09, label="Variation de deltat pour P=0.9 fixé")
ax6.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')

#affichage des figures
fig6.show()

#représentation graphique
ax7.set_title("Variations du rayon en fonction des valeurs de M pour plusieurs valeurs de P fixées")
ax7.set_xlabel("Valeurs de M")
ax7.set_ylabel("Rayon tumoral (en mm)")
ax7.grid(True)
ax7.plot(RXmp06, VARRAYMINP06, label="Variation de rayon pour P=0.6 fixé")
ax7.plot(RXmp06, VARRAYMINP07, label="Variation de rayon pour P=0.7 fixé")
ax7.plot(RXmp06, VARRAYMINP08, label="Variation de rayon pour P=0.8 fixé")
ax7.plot(RXmp06, VARRAYMINP09, label="Variation de rayon pour P=0.9 fixé")
ax7.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')

#affichage des figures
fig7.show()
```

2.3 Étude du paramètre p :

2.3.1 Variations du rayon tumoral pour plusieurs valeurs de p

```
import numpy as np
import matplotlib.pyplot as plt

#paramètres du temps
Tempstot = 20 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance en effet
#de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

#localisations sur les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height


#paramètres de l'espace et autres
#calculer L= taille physique boîte(mm)/
L= 70 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la représentation de l'espace
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion
k=1 #coeff de proliferation
P03=0.65 #proportion de cellules atteintes par la RT
P06=0.85 #proportion de cellules atteintes par la RT
P15=0.95 #proportion de cellules atteintes par la RT
M=2.5 #taux de mort
nbt0 = 950 #appli de la RT
alpha = (D*Delta_t)/dx2 #défini par rapport àl'intervalle de temps, dx et D
print(alpha)
seuil = np.full(N1, 0.07)
seuill = 0.07 #seuil de détection
vitfront = 2*np.sqrt(D*k)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))


#matrice M1D pour les prochains calculs
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
```

```
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
    return MD1

MD1 = MatriceMD1(N1, alpha)
print("MD1= ", MD1)

#on stocke les diagonales dans les nouveaux vecteurs vides
A=np.full(N1, 1+2*alpha)
C=np.full(N1, -alpha)
C[0]=0
B=np.full(N1, -alpha)
B[0]=-2*alpha
B[N1-1]=0

#conditions initiales données par la marche
U01=np.zeros(N1)
U01[0]=1
U01[1]=1

#calcul d'une seule itération pour la diffusion
def resol(B, C, A, R ):
    gam = np.zeros(N1)
    U=np.zeros(N1)
    if (B[0]==0):
        print("ERROR!")
    bet=B[0]
    U[0]= (R[0]/(bet))
    for j in range(1, N1-1):
        gam[j] = C[j-1]/bet
        bet=B[j]-A[j]*gam[j]
        if (bet==0):
            print("ERROR!")
        U[j]=(R[j]-(A[j]*U[j-1]))/(bet)
    k=N1-2
    while k>=0:
        U[k] -= gam[k+1]*U[k+1]
        k = k-1
    gam=np.zeros(N1)
    return U

#boucle pour calculer la diffusion au rang n+1
def solutions(MD1, N1, f):
    if(f==0):
        return U01
    U = np.zeros(N1)
    UI = np.zeros(N1)
    UI = resol(A, B, C, U01)
    if (f==1):
        return UI
```

```
    for i in range(2,nb_iterations+3):
        U = resol(A, B, C, UI)
        UI = U
        if i==(f+1):
            return U

solutionsadt= solutions(MD1, N1, nb_iterations)

#création des vecteurs vides pour la suite

TpreuveTempRT03=[]
TpreuveTempn03=[]
sommem03 = []

TpreuveTempRT06=[]
TpreuveTempn06=[]
sommem06 = []

TpreuveTempRT15=[]
TpreuveTempn15=[]
sommem15 = []

zero = np.zeros(N1)
#fonction calculant la diffusion proliferation avant la RT
def DiffProlif(TpreuveTempn, TpreuveTempRT, U01, nbt0, sommem):
    oo=np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=U01[m] + k*Delta_t*U01[m]*(1-U01[m])
    sommem.extend([oo])
    TpreuveTempn.extend([oo])
    TpreuveTempRT.extend([zero])
    for i in range(1, nbt0):
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- (TP[j]))
        sommem.extend([UP])
        TpreuveTempn.extend([UP])
        TpreuveTempRT.extend([zero])
    return TpreuveTempn, TpreuveTempRT, sommem
TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlif(TpreuveTempn03,
    TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
    diffprolif
TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
    TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
    diffprolif
TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
    TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
    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]
                    >= seuill ) ):
207                 r = (((j-1)+j)/2)*dx
208         RXn.extend([r])
209         RYn.extend([i*Delta_t])
210     return RXn, RYn
211
212 RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
213 RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
214 RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)
215
216 #On cherche a trouver le moment où le rayon est minimal suite a la session
    de RT (deltag) et sa valeur
217 def minn(RX):
218     minim=35
219     indice = 0
220     deltag=0
221     for i in range (nbt0, nb_iterations):
222         m = RX[i]
223         if (m<minim):
224             minim = m
225             indice = i *Delta_t
226             deltag= (i*Delta_t) - (nbt0*Delta_t)
227     return minim, indice, deltag
228
229 minimRX03, indicemin03, deltag03 = minn(RXn03) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
230 print("minimRX", minimRX03)
231 print("indice", indicemin03)
232 print("deltag", deltag03)
233
234
235 minimRX06, indicemin06, deltag06 = minn(RXn06) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
```

```
print("minimRX", minimRX06)
print("indice", indicemin06)
print("deltag", deltag06)

minimRX15, indicemin15, deltag15 = minn(RXn15) #fonction retournant la
    valeur minimum du rayon ainsi que le moment m auquel cette valeur min
    est atteinte
print("minimRX", minimRX15)
print("indice", indicemin15)
print("deltag", deltag15)

#étude du DeltaG
def momentegal(RX):
    bingo=RX[nbt0]
    DELTAG=0
    for i in range(nbt0, nb_iterations):
        if ( ( RX[i-1] <= bingo ) and ( RX[i] >= bingo ) ):
            DELTAG = (i*Delta_t) - (nbt0*Delta_t)
    return DELTAG

DELTAG03 = momentegal(RXn03)
print("DELTAG03 = ",DELTAG03)

DELTAG06 = momentegal(RXn06)
print("DELTAG06 = ",DELTAG06)

DELTAG15 = momentegal(RXn15)
print("DELTAG15 = ",DELTAG15)

#création des figures vides
fig2, ax2 = plt.subplots(ncols=1)
fig2.subplots_adjust(wspace=0.75)

fig4, ax4 = plt.subplots(ncols=1)
fig4.subplots_adjust(wspace=0.75)

#représentation graphique
ax2.set_title(" DIFFUSION PROLIFERATION CELLULES ")
ax2.grid(True)
ax2.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 P=0.9 M=1.1 ans^-1",
       horizontalalignment='right',
       verticalalignment='top',
       transform=ax2.transAxes)
ax2.set_xlabel("Longueur de la boîte (en mm)")
ax2.set_ylabel("Densité cellulaire")
ax2.plot(X, sommem03[0], label="sol_approch c t=0dt", color='black' )
ax2.plot(X, sommem03[20], label="sol_approch c t=20dt", color='black')
ax2.plot(X, sommem03[50], label="sol_approch c t=50dt", color='black')
ax2.plot(X, sommem03[250], label="sol_approch c t=250dt", color='black')
ax2.plot(X, sommem03[500], label="sol_approch ct=500dt moment irradiation RT
    ", color='red')
```

```
ax2.plot(X, sommem03[550], label="sol_approch ct=550dt", color='yellow')
ax2.plot(X, sommem03[625], label="sol_approch ct=625dt", color='orange')
ax2.plot(X, sommem03[750], label="sol_approch c  t=750dt", color='magenta')
ax2.plot(X, sommem03[850], label="sol_approch c  t=850dt", color='purple')
ax2.plot(X, sommem03[950], label="sol_approch c  t=950dt", color='blue')
ax2.plot(X, sommem03[1150], label="sol_approch c  t=1150dt", color='green')
ax2.plot(X, sommem03[1500], label="sol_approch c t=1500dt", color='gray')
ax2.plot(X, sommem03[nb_iterations-1], label="sol_approch c t=Tempstot",
    color='gray')
ax2.legend(bbox_to_anchor=(-0.9, -0.9, -0.9, -0.9), loc='lower left')


ax4.set_title("VARIATION DU RAYON TUMORAL ")
ax4.grid(True)
ax4.set_xlim([0, 18])
ax4.set_ylim([0, 35])
ax4.set_xlabel("Temps (en années)")
ax4.set_ylabel("Rayon de la tumeur (en mm)")
ax4.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1 M=2.5 ans^-1 fixés",
       horizontalalignment='right',
       verticalalignment='top',
       transform=ax4.transAxes)
ax4.annotate('Session de radiothérapie', xy=(9.5,15.1), xytext=(3.5,20.2),
    arrowprops={'facecolor':'black', 'shrink':0.05} )
ax4.plot(RYn03, RXn03, label="P=0.65")
ax4.plot(RYn06, RXn06, label="P=0.85")
ax4.plot(RYn15, RXn15, label="P=0.95")
ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower left')

#affichage des figures

fig2.show()
fig4.show()
```

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

```
import numpy as np
import matplotlib.pyplot as plt

#paramètres du temps
Tempstot = 20 #en années
Delta_t = 0.01 ##en années, il s'agit du pas, on avance
#en effet de 10**-2 années par itération
nb_iterations = int(Tempstot/Delta_t) #taille vecteur temps
print("nbiterations = ", int(nb_iterations))

#paramètres de l'espace et autres
#calculer L= taille physique boite (mm)/
L= 70 #en mm
dx= 0.01 #en mm
dx2 = (dx)**2
N1 = int(L/dx) #taille de la représentation de l'espace
X = np.arange(0 , L, dx)
D = 1 #coeff de diffusion
k=1 #coeff de prolifération
P03=0.65 #proportion des cellules atteintes par la RT
P06=0.85 #proportion des cellules atteintes par la RT
P15=0.95 #proportion des cellules atteintes par la RT
M=2.5 #taux de mort
 #proportion des cellules atteintes par la RT
nbt0 = 950 #Appli de la RT
alpha = (D*Delta_t)/dx2 #defini par rapport àl'intervalle de temps, dx et D
print(alpha)
seuil = np.full(N1, 0.07)
seuill = 0.07 #seuil de détection
vitfront = 2*np.sqrt(D*k)
vitfrontdix = 2*np.sqrt(D*k) + (0.1*2*np.sqrt(D*k))
vit_front=np.full(nb_iterations, 2*np.sqrt(D*k))
vit_frontdix=np.full(nb_iterations, (2*np.sqrt(D*k) +0.1*2*np.sqrt(D*k)))


#matrice M1D pour les prochains calculs
def MatriceMD1(N1, alpha):
    MD1 = np.zeros((N1,N1))
    for i in range (0, N1):
        for j in range (0, N1):
            if (i == j) :
                MD1[i,j]= 1. + 2*alpha
            elif (i+1 == j or i-1 == j) :
                MD1[i,j]= -alpha
    MD1[0,1]= MD1[0,1]-alpha
    return MD1

MD1 = MatriceMD1(N1, alpha)
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,
    TpreuveTempRT03, U01, nbt0, sommem03) #remplissement du vecteur
    diffprolif
135 TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlif(TpreuveTempn06,
    TpreuveTempRT06, U01, nbt0, sommem06) #remplissement du vecteur
    diffprolif
136 TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlif(TpreuveTempn15,
    TpreuveTempRT15, U01, nbt0, sommem15) #remplissement du vecteur
    diffprolif
137
138
139 #fonction calculant les valeurs de la diffusion puis de la prolifération apr
    ès RT
140 def DiffProlifRT(TpreuveTempn, M, TpreuveTempRT, nbt0, sommem, p):
141     oo=np.zeros(N1)
142     op=np.zeros(N1)
143     for i in range(0, N1-1):
144         oo[i]= p*TpreuveTempn[nbt0-1][i]
```

```
        op[i]= (1-p)*TpreuveTempn[nbt0-1][i]

    Tpreuver = resol(A,B,C, oo)
    Tpreuven = resol(A,B,C, op)
    oo = np.zeros(N1)
    op = np.zeros(N1)
    s = np.zeros(N1)
    for m in range(0, N1-1):
        oo[m]=Tpreuver[m] + (M*Delta_t*Tpreuver[m]*(1 - (Tpreuver[m])))
        op[m]= (Tpreuven[m]) + k*Delta_t*Tpreuven[m]*(1 -Tpreuven[m] -
            Tpreuver[m] )
        s[m]= op[m] + oo[m]

    sommem.extend([s])
    TpreuveTempRT.extend([oo])
    TpreuveTempn.extend([op])
    for i in range(nbt0+1, nb_iterations):

        TPRT= np.zeros(N1)
        UPRT=np.zeros(N1)
        TPRT = resol(A,B,C, TpreuveTempRT[i-1])
        s = np.zeros(N1)
        TP= np.zeros(N1)
        UP=np.zeros(N1)
        TP = resol(A,B,C, TpreuveTempn[i-1])
        for j in range(0, N1-1):
            UPRT[j] = TPRT[j] - (M*Delta_t*TPRT[j]*(1-TPRT[j]))
            cd= TPRT[j]
            UP[j] = TP[j] + k*Delta_t*TP[j]*(1- TP[j] - cd)
            s[j] = UP[j] +UPRT[j]

        sommem.extend([s])
        TpreuveTempRT.extend([UPRT])
        TpreuveTempn.extend([UP])
    return TpreuveTempn, TpreuveTempRT, sommem

TpreuveTempn03, TpreuveTempRT03, sommem03 = DiffProlifRT(TpreuveTempn03, M,
    TpreuveTempRT03, nbt0, sommem03, P03)
TpreuveTempn06, TpreuveTempRT06, sommem06 = DiffProlifRT(TpreuveTempn06, M,
    TpreuveTempRT06, nbt0, sommem06, P06)
TpreuveTempn15, TpreuveTempRT15, sommem15 = DiffProlifRT(TpreuveTempn15, M,
    TpreuveTempRT15, nbt0, sommem15, P15)


#Calcul du rayon
RXn03= []
RYn03 = []

RXn06= []
RYn06 = []

```

```
RXn15= []
RYn15 = []

def rayonn(TpreuveTempn, seuill, RXn, RYn):
    for i in range(0, nb_iterations):
        r=0
        for j in range(1, N1-1):
            if ( ( TpreuveTempn[i][j] <= seuill ) and ( TpreuveTempn[i][j-1]
                >= seuill ) ):
                r = (((j-1)+j)/2)*dx
        RXn.extend([r])
        RYn.extend([i*Delta_t])
    return RXn, RYn

RXn03, RYn03 = rayonn(sommem03, seuill, RXn03, RYn03)
RXn06, RYn06 = rayonn(sommem06, seuill, RXn06, RYn06)
RXn15, RYn15 = rayonn(sommem15, seuill, RXn15, RYn15)

#calcul de la pente associée a la courbe du rayon tumoral
def pente(RY, RX):
    pente = np.zeros(nb_iterations)
    for i in range(4, nb_iterations-5):
        pente[i] = (RX[i+4] - RX[i-4]) / (RY[i+4] - RY[i-4])
    return pente
pente03 = pente(RYn03, RXn03)
max_value03 = max(pente03)
maxi03 = (np.where(pente03 == max_value03)[0][0])*Delta_t
print("maxi03", maxi03)

pente06 = pente(RYn06, RXn06)
pente06 = pente(RYn06, RXn06)
max_value06 = max(pente06)
maxi06 = (np.where(pente06 == max_value06)[0][0])*Delta_t
print("maxi06", maxi06)

pente15 = pente(RYn15, RXn15)
pente15 = pente(RYn15, RXn15)
max_value15 = max(pente15)
maxi15 = (np.where(pente15 == max_value15)[0][0])*Delta_t
print("maxi15", maxi15)

#création des figures vides
fig4, ax4 = plt.subplots(ncols=1)
fig4.subplots_adjust(wspace=0.75)

#localisation sur les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height

```

```
#représentation graphique
ax4.set_title("VARIATION DE LA PENTE")
ax4.grid(True)
ax4.set_xlim([0, 19])
ax4.set_xlabel("Temps (en années)")
ax4.set_ylabel("Coefficient de la pente")
ax4.text(.8, -0.4, "D=1mm^2/an K=1 ans^-1",
       horizontalalignment='right',
       verticalalignment='top',
       transform=ax4.transAxes)
ax4.plot(RYn03, pente03, label="Pour P=0.65 pour M=2.5")
ax4.plot(RYn06, pente06, label="Pour P=0.85 pour M=2.5")
ax4.plot(RYn15, pente15, label="Pour P=0.95 pour M=2.5")
ax4.legend(bbox_to_anchor=(-0.5, -0.5, -0.5, -0.5), loc='lower left')

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

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

```
import numpy as np
import matplotlib.pyplot as plt

#définition des valeurs calculées auparavant
RXp = [0.6, 0.7, 0.8, 0.9]

#pour M=1.5: définition des valeurs calculées auparavant
VARDELTAGM15 = [0.48, 0.88, 1.38, 2.14]
VARdeltagM15= [0.17, 0.42, 0.65, 1.11]
VARrayM15= [6.105, 5.935, 5.585, 4.665 ]

#pour M=2.5 : définition des valeurs calculées auparavant
VARDELTAGM25 = [0.74, 1.07, 1.49, 2.17]
VARdeltagM25= [0.31, 0.5, 0.65, 0.97]
VARrayM25= [5.885, 5.565, 4.995, 3.525 ]

#pour M=4.0 : définition des valeurs calculées auparavant
VARDELTAGM4 = [0.83, 1.12, 1.51, 2.16]
VARdeltagM4= [0.34, 0.45, 0.59, 0.82]
VARrayM4=[5.595, 5.155, 4.365, 2.085]

#création des figures vides
fig1, ax1 = plt.subplots()
figa, axa = plt.subplots()
fig2, ax2 = plt.subplots()
figb, axb = plt.subplots()
fig3, ax3 = plt.subplots()
figc, axc = plt.subplots()
fig4, ax4 = plt.subplots()
fig4a, ax4a = plt.subplots()
fig5, ax5 = plt.subplots()

#localisation sur les figures
left, width = .9, .2
bottom, height = .1, 1.5
right = left + width
top = bottom + height

#représentation graphique
ax1.set_title("Variations de DELTAG et deltag en fonction des valeurs de p
    pour M=1.5 fixé")
ax1.set_xlabel("Valeurs de P")
ax1.grid(True)
ax1.set_ylabel("Intervalle de temps en années")
ax1.plot(RXp, VARDELTAGM15, label="Variation de DeltaT")
ax1.plot(RXp, VARdeltagM15, label="Variation de deltat")
ax1.legend()

#affichage figures
```

```
fig1.show()

#représentation graphique
axa.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de P pour M=1.5 fixé")
axa.set_xlabel("Valeurs de P")
axa.grid(True)
axa.set_ylabel("Rayon minimal (en mm)")
axa.plot(RXp, VARrayM15)

#affichage figures
figa.show()

#représentation graphique
ax2.set_title("Variations de DELTAG et deltag en fonction des valeurs de p pour M=2.5 fixé")
ax2.set_xlabel("Valeurs de P")
ax2.grid(True)
ax2.set_ylabel("Intervalle de temps en années")
ax2.plot(RXp, VARDELTAGM25, label="Variation de DeltaT")
ax2.plot(RXp, VARdeltagM25, label="Variation de deltat")
ax2.legend()

#affichage figures
fig2.show()

#représentation graphique
axb.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de p pour M=2.5 fixé")
axb.set_xlabel("Valeurs de P")
axb.grid(True)
axb.set_ylabel("Rayon minimal (en mm)")
axb.plot(RXp, VARrayM25)

#affichage des figures
figb.show()

#représentation graphique
ax3.set_title("Variations de DELTAG et deltag en fonction des valeurs de p pour M=4 fixé")
ax3.set_xlabel("Valeurs de P")
ax3.set_ylabel("Intervalle de temps en années")
ax3.grid(True)
ax3.plot(RXp, VARDELTAGM4, label="Variation du DeltaT pour M=4.0 fixé")
ax3.plot(RXp, VARdeltagM4, label="Variation du deltat pour M=4.0 fixé")
ax3.legend()

#affichage des figures
fig3.show()

#représentation graphique
```

```
axc.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de p pour M=4.0 fixées")
axc.set_xlabel("Valeurs de P")
axc.grid(True)
axc.set_ylabel("Rayon minimal (en mm)")
axc.plot(RXp, VARrayM4)

#affichage figures
figc.show()

#représentation graphique
ax4.set_title("Variations de DELTAG fonction des valeurs de p pour plusieurs valeurs de M fixées")
ax4.set_xlabel("Valeurs de P")
ax4.set_ylabel("Intervalle de temps en années")
ax4.grid(True)
ax4.plot(RXp, VARDELTAGM15, label="Variation du DeltaT pour M=1.5 fixé")
ax4.plot(RXp, VARDELTAGM25, label="Variation du DeltaT pour M=2.5 fixé")
ax4.plot(RXp, VARDELTAGM4, label="Variation du DeltaT pour M=4.0 fixé")

#affichage figures
fig4.show()

#représentation graphique
ax4a.set_title("Variations de deltag en fonction des valeurs de p pour plusieurs valeurs de M fixées")
ax4a.set_xlabel("Valeurs de p")
ax4a.set_ylabel("Intervalle de temps en années")
ax4a.grid(True)
ax4a.plot(RXp, VARdeltagM15, label="Variation du deltat pour M=1.5 fixé")
ax4a.plot(RXp, VARdeltagM25, label="Variation du deltat pour M=2.5 fixé")
ax4a.plot(RXp, VARdeltagM4, label="Variation du deltat pour M=4.0 fixé")
ax4a.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')

#affichages figures
fig4a.show()


#représentation graphique
ax5.set_title("Variations de la valeur du rayon minimal en fonction des valeurs de P pour plusieurs valeurs de M fixées" )
ax5.set_xlabel("Valeurs de P")
ax5.grid(True)
ax5.set_ylabel("Rayon minimal (en mm)")
ax5.plot(RXp, VARrayM15, label="Variation du rayon pour M=1.5 fixé")
ax5.plot(RXp, VARrayM25, label="Variation du rayon pour M=2.5 fixé")
ax5.plot(RXp, VARrayM4, label="Variation du rayon pour M=4 fixé")
ax5.legend(bbox_to_anchor=(-0.6, -0.6, -0.6, -0.6), loc='lower left')

#affichage des figures
fig5.show()
```