Distribution des feux disséminés d'un incendie de forêt

Ahmed Chahlaoui

Plan

Définition

Dissémination des feux:
Les feux disséminés sont les foyers d'incendie allumés lorsque des tisons ou des étincelles produits par un incendie sont transportés par le vent de surface au-delà des limites du foyer principal.

Figure: Dissémination des feux

Problématique

Comment se distribuent les feux disséminés d'une incendie de forêt?

Modélisation

Nécessité d'un modèle stochastique:

- le modèle doit prédire un seul aspect de l'incendie
- le modèle doit être stochastique
- le modèle doit être évalué par des méthodes statistiques

Modélisation

Hypothèses du modèle:

- Le terrain de propagation est horizontal de composition équirépartie
- Les interactions feu-atmosphère sont négligées
- La rotation des colonnes de convection est négligée
- régime quasi-stationnaire de la propagation de l'incendie

Modélisation

Données issus de l'observation du milieu et de l'incendie

- Température T_f (généralement entre 1000 K et 1500 K)
- Vitesse de propagation v_f (généralement entre 3m/s et 6m/s)
- vitesse du vent $w(z, t)$
- Composition du milieu

Modélisation

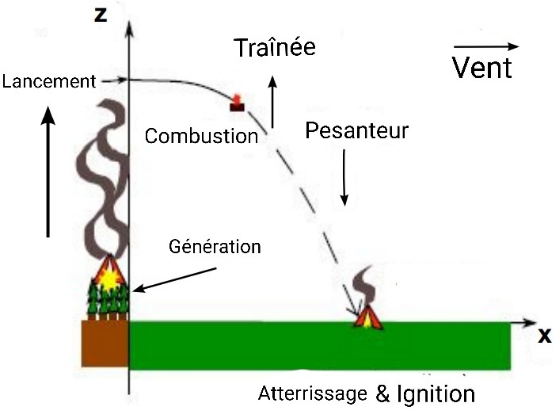


Figure: Schéma de la dissémination des feux les étapes majeurs de la modélisation sont:

1. déterminer une distribution du lancement $\mathcal{L}(z, m)$
2. déterminer une distribution d'atterrissage $\mathcal{A}(x)$
3. déterminer une distribution d'ignition $\mathcal{I}(x)$

- M_0: masse initiale de la particule
- Z_0 : altitude initiale de la particule
- F_0: forme de la particule
- $\mathcal{C}$: composition de la particule en fonction des constituants du milieu

- M_0 à valeurs dans $[0, \infty]$
- On adopte la fonction a densité: $\mu(m) = \alpha e^{-\alpha m}$
- Avec α un paramètre empirique
- Pour $m_1 \leq m_2$ dans $[0, \infty]$:

$$\mathbb{P}(m_1 \leq M_0 \leq m_2) = \int_{m_1}^{m_2} \mu(m) dm$$

Modélisation

Altitude initiale

- On adopte un modèle où Z_0 est indépendante de M_0
- Z_0 à valeurs dans $[0, \infty[$
- On adopte la fonction à densité : $\psi(z) = \lambda e^{-\lambda z}$
- Avec λ un paramètre empirique
- Pour $z_1 \leq z_2$ dans $[0, \infty[$:

$$\mathbb{P}(z_1 \leq Z_0 \leq z_2) = \int_{z_1}^{z_2} \psi(z) dz$$

on adopte la distribution suivante:

$$\mathcal{L}(z, m) = \psi(z)\mu(m)$$

$$\mathbb{P}(z_1 \leq Z_0 \leq z_2, m_1 \leq M_0 \leq m_2) = \int_{z_1}^{z_2} \int_{m_1}^{m_2} \mathcal{L}(z, m) dm\ dz$$

Modélisation

Forme de la particule

On assimile les particules a des formes cylindriques ou sphériques.

$$F_0 = \begin{cases} 1 & \text{si la particule est sphérique} \\ 0 & \text{si la particule est cylindrique} \end{cases}$$

Le coefficient de traînée est donné par:

$$C_d = 0.47F_0 + 0.82(1 - F_0)$$

Supposons que le milieu est composé de N substances avec des titres massiques $(x_i)_{1 \leq i \leq n}$.
La probabilité que la particule soit formée de la matière i est:

$$\mathbb{P}(\mathcal{C} = i) = x_i$$

Pour une grandeur A dépendante du constituant (e.g. masse volumique, capacité calorifique massique,...)

$$A = \sum_{i=1}^{N} a_i L_i(\mathcal{C})$$

- $(a_i)_{1 \leq i \leq N}$ grandeurs associés aux constituants
- $(L_i)_{1 \leq i \leq N}$ suite de polynômes de Lagrange associées a la famille $\{1, 2, ..., N\}$

Modélisation

Étude mécanique de la particule lancée

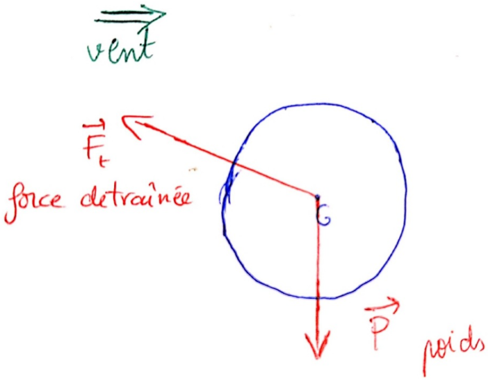


Figure: Bilan des forces agissantes sur la particule dans le référentiel lié au front de l'incendie

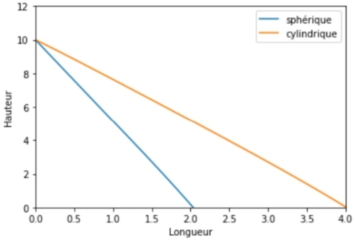
Dans le référentiel galiléen lié au front de l'incendie, on applique le principe fondamental de la dynamique:

$$\frac{d\overrightarrow{p(t)}}{dt} = m(t)\overrightarrow{g} - \frac{1}{2}\rho C_d A(t) \mid\mid \overrightarrow{v(t)} - \overrightarrow{w(z,t)} + \overrightarrow{v_f} \mid\mid (\overrightarrow{v(t)} - \overrightarrow{w(z,t)} + \overrightarrow{v_f})$$

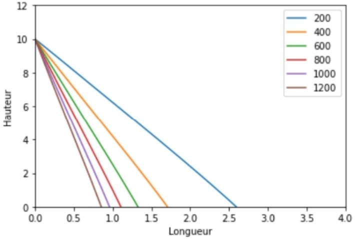
Modélisation

Étude mécanique de la particule lancée

- la masse de la particule: $m(t)$
- la surface de la particule $A(t)$
- la masse volumique de la particule ρ
- le coefficient de traînée C_d
- la vitesse du vent $\overrightarrow{w}(z, t)$ par rapport au référentiel terrestre



(a) La forme



(b) La masse volumique

Figure: Nécessité de l'introduction des variables aléatoires

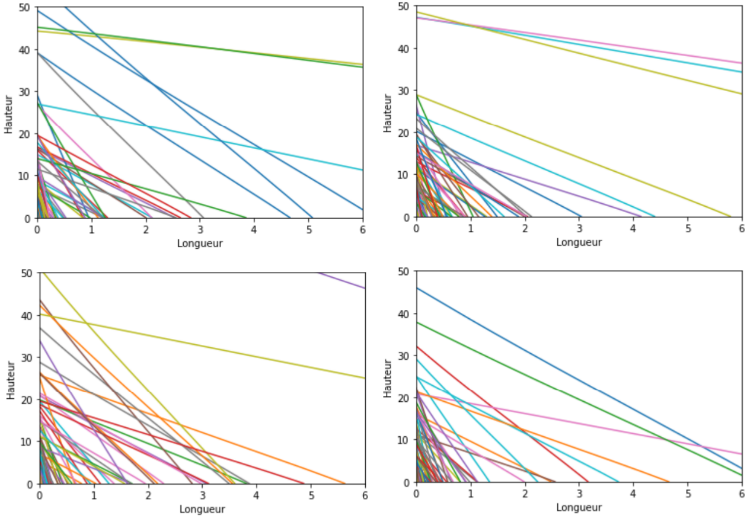


Figure: Simulation avec 100 particules

Modélisation

Étude thermodynamique sur la particule lancée

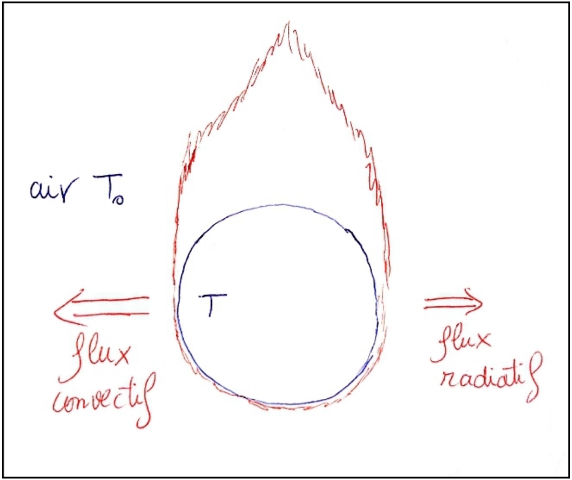


Figure: Bilan thermique sur la la particule lancée

La particule lancée, lors de sa chute, est soumise aux flux convectifs et radiatifs.
D'après le premier principe de la thermodynamique:

$$m(t)C_p\frac{dT}{dt} = -A(t)(\underbrace{\sigma\epsilon(T^4 - T_0^4)}_{radiatif} + \underbrace{h(T - T_0)}_{convectif})$$

Étude thermodynamique sur la particule lancée

- C_p: capacité calorifique massique
- σ : constante de Stefan-Boltzmann ($5.67.10^{-8} Wm^{-2}K^{-4}$)
- ϵ: émissivité (entre 0 et 1)
- h: coefficient de transfert thermique
- T_0: Température ambiante

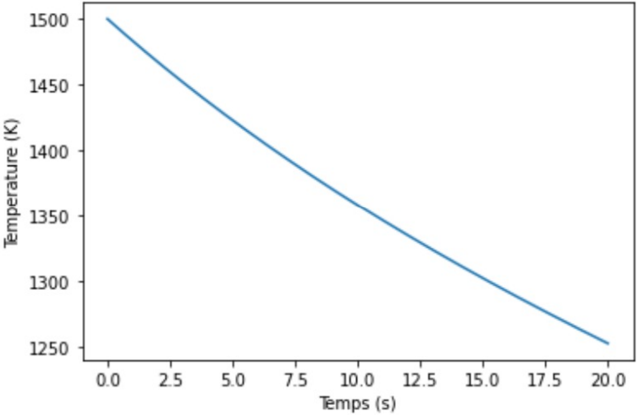


Figure: Évolution de la température de la particule lancée

Modélisation

Équilibre thermodynamique avec le sol

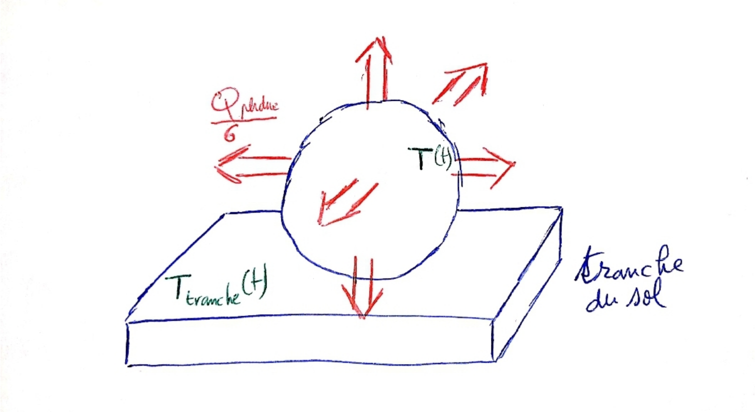


Figure: Bilan thermique sur la tranche du sol

Modélisation

Équilibre thermodynamique avec le sol

- Par isotropie, Une tranche du sol reçoit $\frac{1}{6}$ de la chaleur émise par la particule.
  $\forall t \geq t_{arrivée}$

$$m_{tranche} c_{sol} \frac{dT_{tranche}}{dt} = -\frac{1}{5} m(t) Cp \frac{dT}{dt}$$

- Il y a ignition si $\exists t \geq t_{arrivée} \quad T_{tranche}(t) \geq T_{ig}$

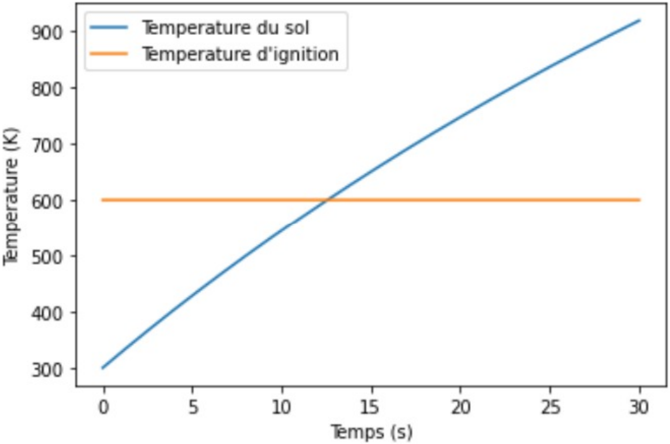


Figure: Ignition du sol

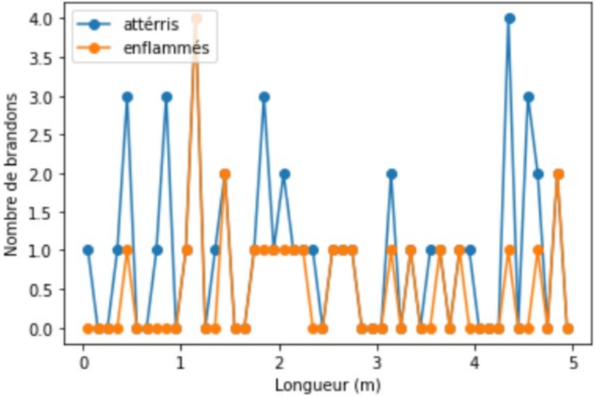
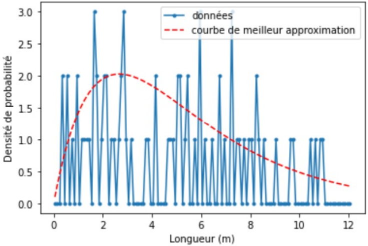
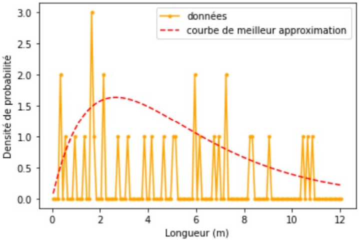


Figure: Atterrissage et Ignition



(a) L'atterrissage $y = 2.06xe^{-0.37x}$



(b) L'ignition $y = 1.68xe^{-0.37x}$

Expérience

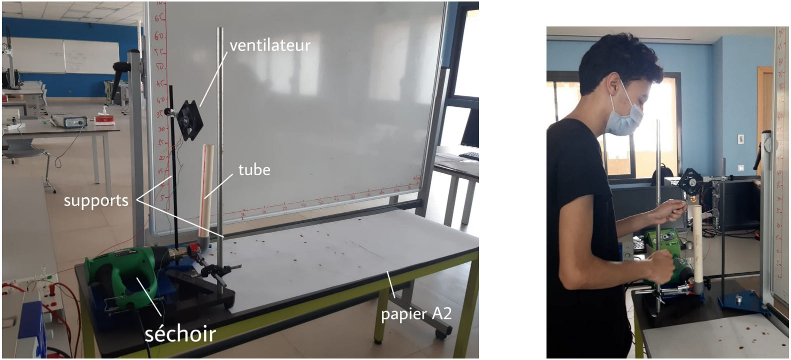


Figure: Montage expérimental

Expérience

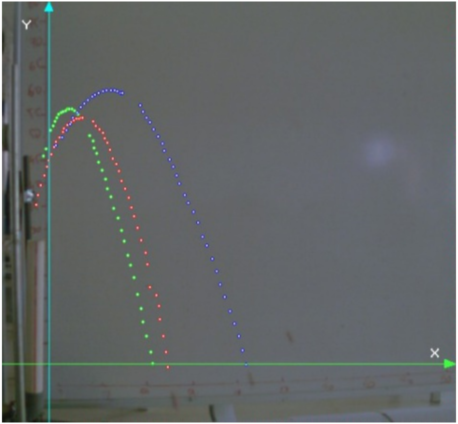


Figure: Trajectoire des particules

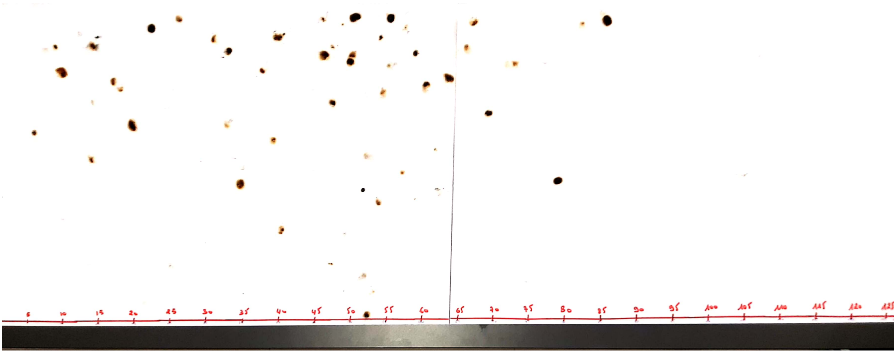


Figure: dissémination de 50 particules

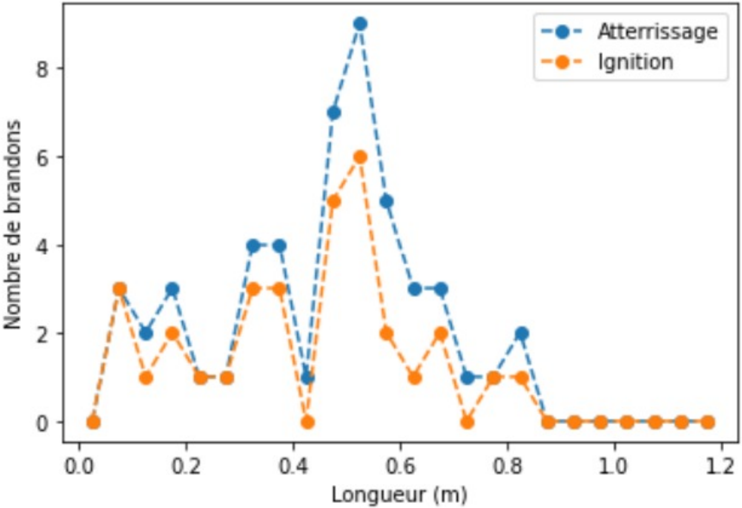


Figure: Résultats expérimentaux

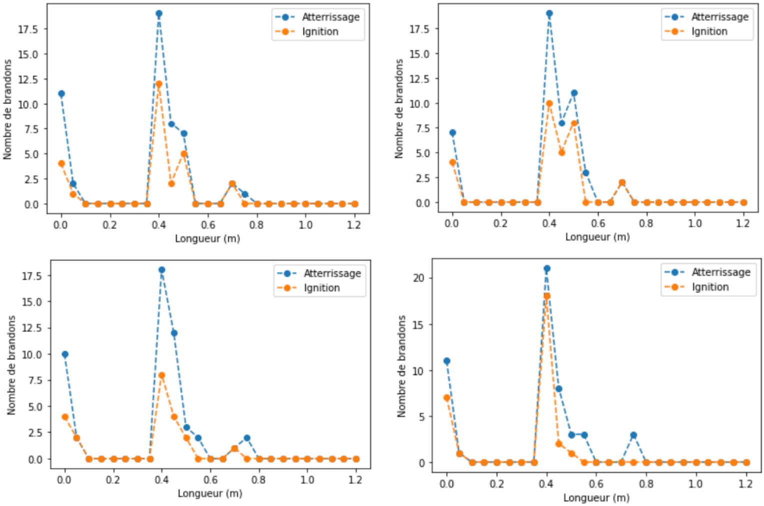


Figure: Résultats théoriques avec 50 particules

Conclusion

- Résultats expérimentaux et simulation
- Avantages du modèle
- Inconvénients du modèle

Annexe

Simulation

```
from scipy.integrate import odeint
from pylab import *
from random import *
#constantes
g = 9.8   #acceleration de la pesanteur
f = 0.0001 #taux de combustion
p = 1.225 #masse volumique de l'air
T0 = 300 #Temperature ambiante
Cp = 1400 #capacite thermique massique du papier
s = 5.67*10**(-8) #constante de steffan boltzmann
e = 0.9 #émissivité
h = 5 #coefficient de transfert thermique
T0=300 #Temperature ambiante
Tig = 600 #Temperature d'ignition du gazon
v_f = 1 #vitesse de propagation de l'incendie
L=[200, 600, 1000, 1300, 1800, 2200] #masses volumiques
Cp = [10,100,200,500,1000,2000] #capacites thermiques massiques
M = [[0.1*i+0.05,0,0] for i in range(1000)]
PP1, PP2 = [], []
mtot=0
for l in range(100):
    #masse initiale
    m0 = expovariate(1000)
    #altitude initiale
    z0 = expovariate(0.1)
    #Composition
    C0 = randrange(6)
    r = L[C0]
    Cp = L[C0]
    #Forme
    F0 = randrange(0,2)
    #profile du vent quadratique
```

Annexe

Simulation

```
    def wind(y):
        return wxref*((y/yref)**k)
    #position initiale
    p0=[0,z0]
    x = [p0[0]]
    y = [p0[1]]
    # vitesse initiale
    v0 = [2,0]
    vx=[v0[0]]
    vy=[v0[1]]
    w=[wind(p0[1]),wyref]
    #masse du brandon
    K1 = f*(4*pi*r)**(1/3)*(3**(-1/3))
    K2 = f*m0**(2/3)*(r*pi/100)**(1/3)
    def m(t):
        return (m0-K2*t)*(1-F0)+(m0**(1/3)-K1*t)**(1/3)*F0
    #derivee de la masse
    def dm(t):
        return -K2*(1-F0)-K1*F0*3*(m0**(1/3)-K1*t)**2
    #surface du brandon
    A0 = (pi/5*((100*m0)/(pi*r))**(2/3))*(1-F0)+4*pi*(((3*m0)/(4*pi*r))**(2/3))*(F0)
    def A(t):
        return A0*(m(t)/m0)*(1-F0)+F0*A0*(m(t)/m0)**(2/3)
    #coefficient de trainee
    Cd = 0.49*F0+0.82*(1-F0)
    #Resolution de l'equation differentielle du mouvement
    def fx(x,y,t):
        return 1/m(t)*(-dm(t)*x-(0.5*p*Cd*A(t)*((x-w[0]+v_f)**2+(y-w[1])**2)**0.5)*(x-w[0]+v_f))
    def fy(x,y,t):
        return 1/m(t)*(-dm(t)*y-m(t)*g-(0.5*p*Cd*A(t)*((x-w[0]+v_f)**2+(y-w[1])**2)**0.5*(y-w[1])))
    for i in range(0,1000):
        t=linspace(0.1*i,0.1*(i+1),100)
        vxi=vx[len(vx)-1]
        vyi=vy[len(vy)-1]
        for j in range(1,len(t)):
            if m(t[j]) <= 0.00001 :
                break
```

Annexe

Simulation

```
        a=(t[j]-t[j-1])*(fx(vxi,vyi,t[j]))+vxi
        b=(t[j]-t[j-1])*(fy(vxi,vyi,t[j]))+vyi
        vx.append(a)
        vy.append(b)
        c=x[len(x)-1]+a*(t[j]-t[j-1])+0.5*fx(vxi,vyi,t[j])*((t[j]-t[j-1])**2)
        d=y[len(y)-1]+b*(t[j]-t[j-1])+0.5*fy(vxi,vyi,t[j])*((t[j]-t[j-1])**2)
        if d<0:
            tt=t[j]
            break
        x.append(c)
        y.append(d)
        vxi=a
        vyi=b
    if m(t[j]) <= 0.00001 :
        break
    if d<0:
        tatt=t[j]
        break
    w[0]=wind(y[len(y)-1])
Bb= [tatt,x[-1]] #temps et position d'atterrissage
#Resolution des equations differentielles thermodynamiques
def model(T,t):
    dTdt = -(A(t)/(m(t)*Cp))*(s*e*(T**4-T0**4)+h*(T-T0))
    return dTdt
t = linspace(0,20,20000)
Ti=1500 #Temperature initiale du debris (de l'incendie aussi)
H =odeint(model,Ti,t) #Temperature du debris lancee
msol=1
csol = 0.00016 #capacite thermique massique du gazon
def Tdebris(t):
    n= int(t/0.002)
    return H[n][0]

Tdebrislist=[Tdebris(l) for l in t[:9001]]
```

Annexe

Simulation

```python
    def model2(Ts,t):
        dTsdt= A(t+tatt)/(msol*csol*5)*(s*e*(Tdebris(t+tatt)**4-T0**4)+h*(Tdebris(t+tatt)-T0))
        return dTsdt

    H2= odeint(model2, T0, t) #Temperature de la tranche du sol
    print(Bb)
    plot(x,y)
    PP1.append(x)
    PP2.append(y)
    b = int(x[-1]/.1)
    mtot += m(tatt)
    if b>= 0 and b<1000 :
        M[b][1] += 1
        if H2[-1][0] > Tig :
            M[b][2] +=1
    print(H2[-1][0])

#Plotting
plot(x,y)
axis([0,6,0,50])
grid(False)
xlabel('Longueur')
ylabel('Hauteur')
print(M)
show()
Mu = [M[i][0] for i in range(100)] #positions d'atterrissage
Su = [M[i][1] for i in range(100)] #nombre de brandons atterris
Tu = [M[i][2] for i in range(100)] #nombre de brandons enflammes
plot(Mu,Tu,linestyle = '-', marker = 'o')
xlabel('Longueur (m)')
ylabel('Nombre de brandons')
legend(['attérris','enflammés'],loc='upper left')
show()
```

Annexe

simulation des résultats expérimentaux

```
from scipy.integrate import odeint
from pylab import *
from random import *
#constantes
g = 9.8   #acceleration de la pesanteur
f = 0.0001 #taux de combustion
p = 1.225 #masse volumique de l'air
T0 = 300 #Temperature ambiante
Cp = 1400 #capacite thermique massique du papier
s = 5.67*10**(-8) #constante de steffan boltzmann
e = 0.9 #émissivité
h = 5 #coefficient de transfert thermique
r = 1200 #masse volumique du papier

M = [[i,0,0] for i in range(25)] #data
for l in range(50):
    #masse initiale
    MM=[.00005,0.0001,0.00015,0.0002,0.00025,0.0003,0.00035]
    mrand= randrange(6)
    m0=MM[mrand]
    #altitude initiale
    z0 = 0.3+expovariate(11.5)
    #Forme
    F0 = 1
    #profile du vent
    def wind(y):
        if abs(y-0.35)<= 0.1 :
            return wxref
        return 0
    #position initiale
```

Annexe

simulation des résultats expérimentaux

```python
    vy=[v0[1]]
    w=[wind(p0[1]),wyref]
    #masse du brandon
    K1 = f*(4*pi*r)**(1/3)*(3**(-1/3))
    K2 = f*m0**(2/3)*(r*pi/100)**(1/3)
    def m(t):
        return (m0-K2*t)*(1-F0)+(m0**(1/3)-K1*t)**(1/3)*F0
    def dm(t):
        return -K2*(1-F0)-K1*F0*3*(m0**(1/3)-K1*t)**2
    A0 = (pi/5*((100*m0)/(pi*r))**(2/3))*(1-F0)+4*pi*(((3*m0)/(4*pi*r))**(2/3))*(F0)
    #surface du brandon
    def A(t):
        return A0*(m(t)/m0)*(1-F0)+F0*A0*(m(t)/m0)**(2/3)
    #coefficient de trainee
    Cd = 0.49*F0+0.82*(1-F0)
    #Resolution de l'equation differentielle du mouvement
    def fx(x,y,t):
        return 1/m(t)*(-dm(t)*x-(0.5*p*Cd*A(t)*((x-w[0]+v_f)**2+(y-w[1])**2)**0.5)*(x-w[0]+v_f))
    def fy(x,y,t):
        return 1/m(t)*(-dm(t)*y-m(t)*g-(0.5*p*Cd*A(t)*((x-w[0]+v_f)**2+(y-w[1])**2)**0.5*(y-w[1])))
    for i in range(0,1000):
        t=linspace(0.1*i,0.1*(i+1),100)
        vxi=vx[len(vx)-1]
        vyi=vy[len(vy)-1]
        for j in range(1,len(t)):
            if m(t[j]) <= 0.00001 :
                break
            a=(t[j]-t[j-1])*(fx(vxi,vyi,t[j]))+vxi
            b=(t[j]-t[j-1])*(fy(vxi,vyi,t[j]))+vyi
            vx.append(a)
            vy.append(b)
            c=x[len(x)-1]+a*(t[j]-t[j-1])+0.5*fx(vxi,vyi,t[j])*((t[j]-t[j-1])**2)
            d=y[len(y)-1]+b*(t[j]-t[j-1])+0.5*fy(vxi,vyi,t[j])*((t[j]-t[j-1])**2)
            if d<0:
                tt=t[j]
                break
```

Annexe

simulation des résultats expérimentaux

```
            x.append(c)
            y.append(d)
            vxi=a
            vyi=b
        if m(t[j]) <= 0.00001 :
            break
        if d<0:
            tatt=t[j]
            break
        w[0]=wind(y[len(y)-1])
    Bb= [tatt,x[-1]] #temps et abscisse d'atterrissage
    #Resolution des equation differentielles thermodynamiques
    def model(T,t):
        dTdt = -(A(t)/(m(t)*Cp))*(s*e*(T**4-T0**4)+h*(T-T0))
        return dTdt
    t = linspace(0,20,20000)
    Ti=1500 #Temperature du brandon initiale
    #Temperature du brandon
    H =odeint(model,Ti,t)

    msol=1
    csol = 1400 #sol du papier
    def Tdebris(t):
        n= int(t/0.002)
        return H[n][0]
    Tdebrislist=[Tdebris(l) for l in t[:9001]]

    def model2(Ts,t):
        dTsdt= A(t+tatt)/(msol*csol*5)*(s*e*(Tdebris(t+tatt)**4-T0**4)+h*(Tdebris(t+tatt)-T0))
        return dTsdt
    #Temperature du sol
    H2= odeint(model2, T0, t)

    print(Bb)
    plot(x,y)
    b = int(x[-1]/.05)
    mtot += m(tatt)
```

Annexe

simulation des résultats expérimentaux

```python
    if b>= 0 and b<1000 :
        M[b][1] += 1
        if H2[-1][0] > 600 :
            M[b][2] +=1
    print(H2[-1][0])

#Plotting
axis([0,1,0,1])
xlabel('Longueur')
ylabel('Hauteur')
print(M)
show()
Mu = [M[i][0]*0.05 for i in range(len(M))] #positions d'atterrissage
Su = [M[i][1] for i in range(len(M))]      #nombre de brandons atterris
Tu = [M[i][2] for i in range(len(M))]      #nombre de brandons enflammes
plot(Mu,Su,linestyle = '--', marker = 'o')
plot(Mu,Tu,linestyle = '--', marker = 'o')
xlabel('Longueur (m)')
ylabel('Nombre de brandons')
legend(['Atterrissage','Ignition'],loc='upper right')
show()
```

Annexe

Curve fitting

```python
import numpy as np
from scipy.optimize import curve_fit
from matplotlib import pyplot
#meilleur approximation de la forme
def objective(t, a, b, c):
    return a*t**int(abs(c)) * np.exp(-b * t)

# les donnees
y0 = [[0.05, 0, 0], [0.15000000000000002, 0, 0], [0.25, 0, 0], [0.35000000000000003, 2, 2], [0.45, 0, 0],
y=[y0[i][1] for i in range(len(y0))]
x= [0.1*i+0.05 for i in range(len(y))]

# curve fit
popt, _ = curve_fit(objective, x, y, maxfev=5000)
# summarize the parameter values
a, b, c = popt
print(popt)
# plot input vs output
pyplot.plot(x,y,linestyle='-',color='b',marker='.',label='données')
x_line = np.arange(min(x), max(x), 1)
y_line = objective(x_line, a, b, c)
# create a line plot for the mapping function
G = [objective(t,a,b,c)    for t in x ]
pyplot.plot(x,G,linestyle='--',color='red',label='courbe de meilleur approximation')
pyplot.xlabel('Longueur (m)')
pyplot.ylabel('Densité de probabilité')
pyplot.legend()
pyplot.show()
```