Étude et modélisation de la propagation d'un feu de forêt

Antonin Cazals - 43278

CPGE MP

Plan

- Modélisation informatique de la propagation d'un incendie en milieu forestier
- Modélisation physique de la propagation d'un incendie en milieu forestier

Modélisation Informatique

Hypothèses de modélisation

- Discrétisation du temps et de l'espace
- Structure de la forêt : uniforme, deux dimensions
- Structure du feu : propabilité de propagation dans 4 directions
- Combustion Complète

Modélisation Informatique

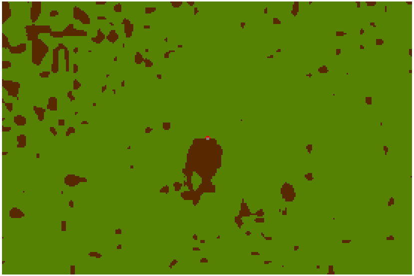
Modèle de l'automate cellulaire : 1 arbre = 1 cellule

- matrice d'états de la forêt
- 3 états possibles : (0, arbre en vie) ; (1, arbre en feu) ; (2, arbre carbonisé)
- $e_{i,j}$ état de la cellule
- $k_{i,j}$: itération ou l'arbre en (i,j) brule, -1 si l'arbre est en vie

$$\begin{pmatrix} e_{11}, k_{11} & e_{12}, k_{12} & \dots & e_{1n}, k_{1n} \\ e_{21}, k_{21} & e_{22}, k_{22} & \dots & e_{2n}, k_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ e_{n1}, k_{n1} & e_{n2}, k_{n2} & \dots & e_{nn}, k_{nn} \end{pmatrix}$$

Modélisation Informatique

- Traitement d'Image : Conversion en automate

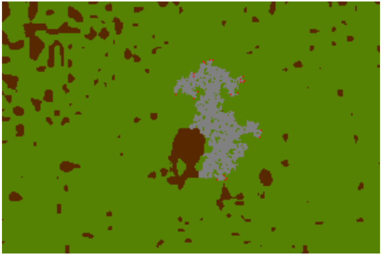



Paramètres de la simulation

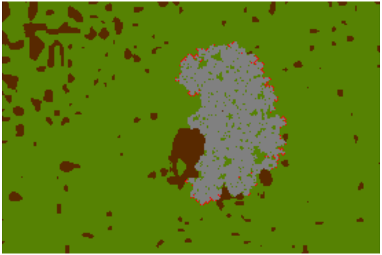
1. Densité de la forêt (élagage)
2. Vent
3. Durée de combustion d'un arbre
4. Propabilité de propagation du feu

Modélisation Informatique

- forêt non élaguée, pas de vent,
- T_c=1 itération
- 100 itérations

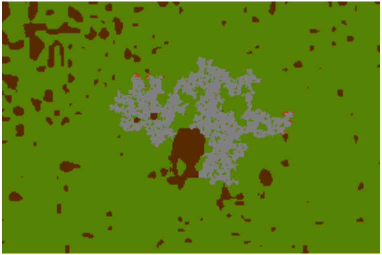


$p = 0.49$

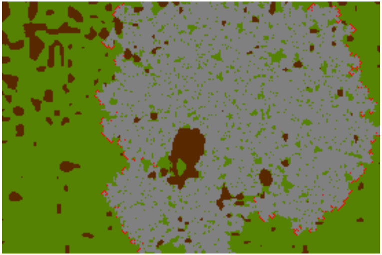


$p = 0.51$

- 200 itérations

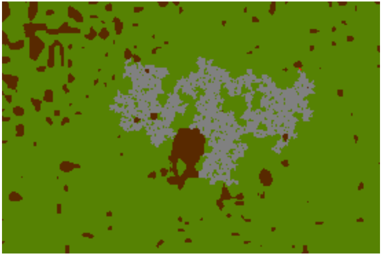


$p = 0.49$

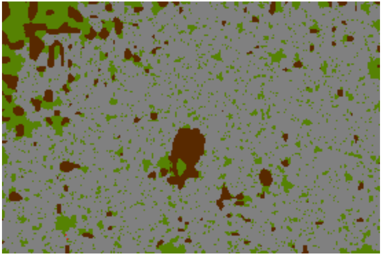


$p = 0.51$

- Dernière Itération



$p = 0.49$



$p = 0.51$

- Proportion de forêt Brulée : 22% ; 91%

Phénomène de Percolation

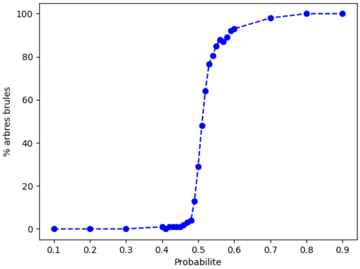
Définition

1. système = réseau de sites en liens
2. differents états, transmission de l'information aux voisins
3. phénomène de seuil associé à la transmission d'une information

Seuil de Percolation

1. p $>$ $P_c \Rightarrow$ Percolation
2. p $<$ $P_c \Rightarrow$ Pas de Percolation
3. p $=$ $P_c \Rightarrow$ Instable

Percolation : déterminer expérimentalement le seuil de percolation

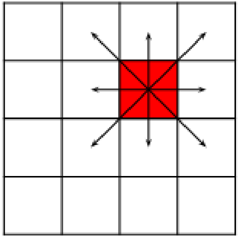


- Théorie : $P_c = 0.5$
- Expérimentalement : $P_c \simeq 0.5$

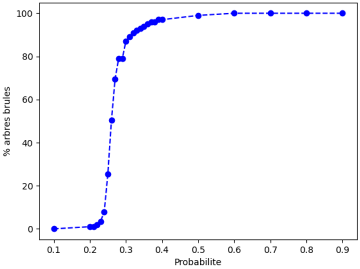
Complexification de la modélisation

Propagation

1. Propagation dans 8 directions
2. Résultat Théorique pour 4 voisins plus valable
3. Détermination expérimentale du seuil de propagation pour une propagation plus complexe



Propagation dans 8 directions



- Expérimentalement : $P_c \simeq 0.25$

Extraire des Informations

1. évolution d'états d'une case, V,R,N : $X_{i,j,n}$
2. nombre de cases vertes à k :
$$V_k$$
3. nombre de nouvelles cases en feu à k :
$$F_k$$
4. nombre de cases vertes atteignables par le feu à l'itération k :
$$N_k$$
5. Mesure de P_p : déterminer
$$P(X_{i,j,k+1} = R | X_{i,j,k} = V)$$

Modélisation Informatique

- exemple pour un cas à 2 itérations :
- k=0, E_0 :V_0= 10 et $F_0 = 0$
- k=1, E_1 :V_1= 8 et F_1= 2
- k=2, E_2 :$V_2 = 5$; $F_2 = 3$
- Fonction de vraisemblance :

$$j(p) = P([V_2 = 5, F_2 = 3] \cap [V_1 = 8, F_1 = 2])$$

- Loi des probabilités composées :

$$P(E_0 \cap E_1 \cap E_2) = P(E_2|E_1 \cap E_0) \cdot P(E_1|E_0) \cdot P(E_0)$$

Modélisation Informatique

- case dans N_k à l'itération K : probabilité P_p de devenir rouge à k+1 $\Rightarrow$ Loi de Bernouilli, paramètre P_p
- Ensemble des cases de l'espace à l'itération k : loi binomiale (n= N_k, p $=P_p$)

Vraisemblance

$$j(P_p) = \prod_{k=1}^{N} \binom{N_k}{F_{k+1}} \cdot {P_p}^{F_{k+1}} \cdot (1 - P_p)^{N_k - F_{k+1}}$$

Modélisation Informatique

Objectif : Estimation au sens du maximum de vraisemblance P_p

$$\log(j(P_p)) = \sum_{k=1}^{N} \log \binom{N_k}{F_{k+1}} + F_{k+1}\log(P_p) + (N_k - F_{k+1})\log(1-P_p)$$

$$\frac{\partial \log(j(P_p))}{\partial P_p} = \sum_{k=1}^{N} \frac{F_{k+1}}{P_p} - \frac{N_k - F_{k+1}}{1-P_p}$$

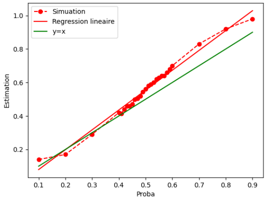
$$\frac{\partial \log(j(P_p))}{\partial P_p} = 0$$

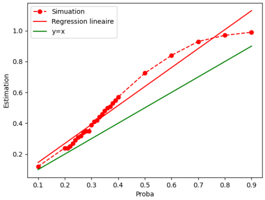
$\Rightarrow$

$$\hat{P}_p = \frac{\sum_{k=1}^{N} F_{k+1}}{\sum_{k=1}^{N} N_k}$$

Résultas de l'estimation

- Médiane de $\hat{P}_p$ sur 50 simulations

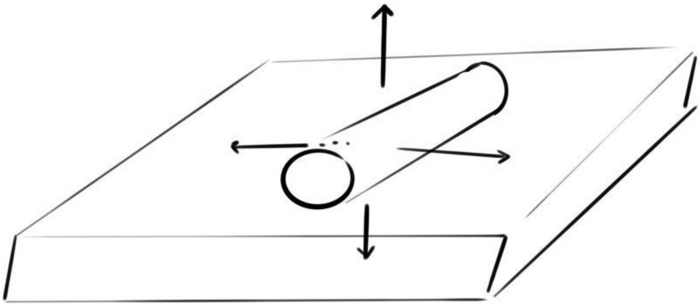




Foret Initiale $y = 1.18$, $r^2 = 0.98$ Foret élaguée $y = 1.29$, $r^2 = 0.98$

Objectifs

1. Modéliser la propagation du feu à une échelle Locale
2. Tige enflammée sur le sol



Modélisation Physique

- cylindre : r(t), h(t)
- Hypothèse : r(t) $\sim$ *constante* $=$r$_0$

$$\frac{dh(t)}{dt} = -C$$

$$h(t) = h_0 - C \cdot t$$

$$A(t) = 2 \cdot \pi \cdot r_0 \cdot (h(t) + r_0)$$

$$m(t) = m_0 - k \cdot t$$

$$k = \rho \cdot \pi \cdot r_0^2 \cdot k$$

Modélisation Physique

- Flux Radiatif
- Flux Convectif
- Loi de Stefan, Loi de Newton :

$$m(t) \cdot c_p \cdot \frac{dT}{dt} = -A(t)(\sigma \cdot (T^4 - T_0^4) + h(T - T_0)) \tag{1}$$

- c_p : capacité calorifique massique
- σ : constante de Boltzmann
- h : coefficient de transfert thermique
- T_0 : Température Ambiante

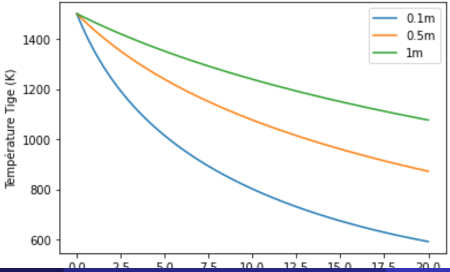
Modélisation Physique

- Température d'ignition des feuillages : $T_i = 505$ K
- Chaleur échangée : isotrope

$$dH_{\text{portion}} = m_{\text{portion}} \cdot c_{\text{sol}} \cdot dT_{\text{portion}} = \frac{1}{4} \cdot \delta Q_{\text{perdue}} = -\frac{1}{4} \cdot dH_{\text{tige}} \tag{2}$$

$$m_{\text{portion}} \cdot c_{\text{sol}} \cdot \frac{dT_{\text{portion}}}{dt} = -\frac{1}{4} \cdot m(t) \cdot C_p \cdot \frac{dT}{dt} \tag{3}$$

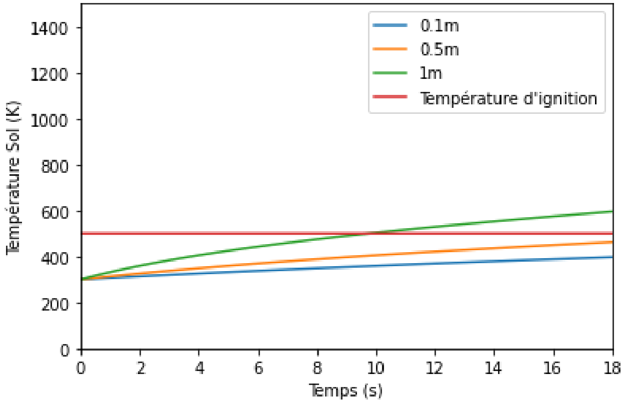
- Température de la tige
- $h = 5 \frac{W}{m^2 K}$: Coefficient de transfert thermique
- $T_0 = 300K$: Température de référence
- $T_{\text{initiale}} = 1500K$: Température initiale
- $c_{\text{sol}} = 0.001 \frac{J}{kgK}$: Capacité thermique massique du sol
- $m_{\text{tranche}} = 1kg$: Masse de la tranche de sol considérée



Courbes Théoriques

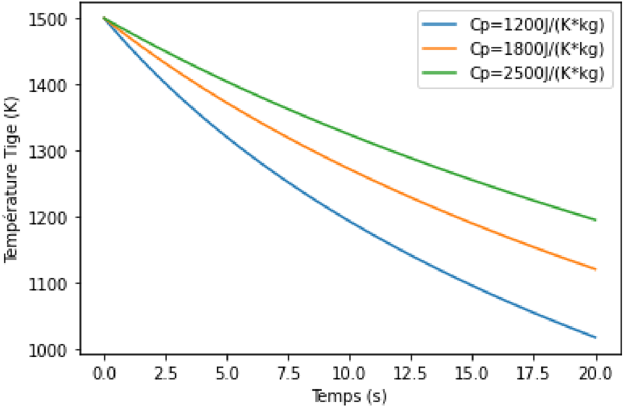
- Équations différentielles non linéaires $\Rightarrow$ Résolution Numérique

- Température du sol



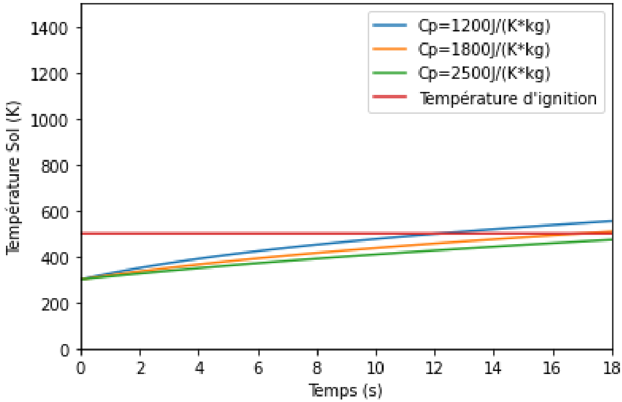
Courbes Théoriques

- Équations différentielles non linéaires $\Rightarrow$ Résolution Numérique
- Température de la tige



Courbes Théoriques

- Équations différentielles non linéaires $\Rightarrow$ Résolution Numérique
- Température du sol



Conclusion

1. Bilan :
2. échelle globale
3. phénomènes de percolation
4. échelle locale

- Ce que le TIPE m'a apporté :
- Progresser en programmation
- Apprendre à écrire en Latex
- Étudier des documents scientifiques
- Apprendre à développer un modèle scientifique

```python
from PIL import Image
from numpy import asarray
import numpy as np
#import random
import argparse
import sys
import random
import os
import re
import matplotlib.pyplot as plt
import statistics
from scipy import stats


# RGB colors
rgb_alive = [86, 130, 3]
rgb_notree = [88, 41, 0]
rgb_burning = [255,0,0]
rgb_burned = [128,128,128]

# labels used for trees
alive = 0
burning = 1
burned = 2
notree = 3

# Convert an image into an RGB matrix, with 2 colors: one for trees, one for the
# remaining pixels
def get_rgb_matrix(filename):
  img = Image.open(filename)

  t = asarray(img)
  n = len(t)
  m = len(t[0])
  I = np.zeros((n,m,3), dtype="uint8") # creates a  matrix n x m with triples of zeroes

  # for each pixel, if g dominates: creates a green cell; red otherwise
  for i in range (n):
    for j in range(m):
       if (t[i][j][1]>t[i][j][0] and t[i][j][1]>t[i][j][2]):
            I[i][j] = rgb_alive
       else :
         I[i][j] = rgb_notree
  return I


# get the offsets (-1, 0, +1) of the neighbors of the cell (i,j) in matrix M
# NB: topology==4 or 8, that is 4 or 8 neighbors
# NB: a list of pairs (a,b) is returned
def get_neighbors_offsets(M,i,j, topology):
```

```
def get_neighbors_offsets(M,i,j, topology):
  L = []
  n = len (M)
  m = len (M[0])
  # (-1,0,1): offset with repect to i; likewise for j
  for a in range(-1,2):
    for b in range(-1,2):
      # select the four neighbors above/below/left/right; remove center  + corners
      if 0<=i+a<=n-1 and 0<=j+b<=m-1:
        if topology == 4:
          if not ((a==0 and b==0) or (a== -1 and b==-1) or (a==-1 and b==1) or (a==1 and b==-1) or (a==1 and 
            L.append((a, b))
        # anto: a toi de jour
        elif topology == 8:
         if not (a==0 and b==0):
          L.append((a, b))
        else:
          sys.exit("Connectivity must be 4 or 8")
  return(L)

# if voxel (i,j) is not burning: return empty list
def alive_neighbors_offsets_of_burning_voxel(M,i,j, topology):
  if M[i][j][0] != burning:
    return []

  L = get_neighbors_offsets(M, i,j, topology) # neighbor: pair of indices [i,j]
  P=[]
  for k in range(len(L)):
    # coords of the neighbors
    a=L[k][0];  b=L[k][1]
    if M[i+a][j+b][0] == alive:
      P.append(L[k])
  return P

## NB: neighbors presented as tuples (i,j) not lists [i,j], to make them hashable in sets
#i##############################################################################
class Fire_simulation:

  # create the forest matrix M; set the simulation parameters
  def __init__(self, options):
    self.options = options

    self.num_trees_total = 0
    self.wind = [0, 0] # wind: default is no wind

    self.ofname_prefix = "" # prefix used for the output files
    self.dfname_stats = "" # filename to dump stats
    self.dfname_pict = ""  # filename to dump images

    self.n_hat_estimation = [] #list of pairs of integers to estimate n: see below
```

```python
    self.p_hat_estimation = [] #list of pairs of integers to estimate p: see below

  # prepares self.M_forest, and counts the total number of trees
  def get_forest_matrix(self, rgb_matrix):
    self.num_trees_total = 0
    n = len(rgb_matrix)
    m = len(rgb_matrix[0])
    #self.M_forest = np.zeros((n, m, 2), dtype="uint8")
    self.M_forest = np.zeros((n, m, 2), dtype="int16")

    for i in range(n):
      for j in range (m):
        keep_tree = True
        a = random.randint(0,100) # percentage
        if a < 100*self.options.trim_proba:
          keep_tree = False

        if rgb_matrix[i][j][0] == rgb_alive[0] and keep_tree:
          self.M_forest[i][j] = [alive,0]
          self.num_trees_total += 1
        else:
          self.M_forest[i][j] = [notree, 0]

    print("Image n m size num-trees:", n, m, n*m, self.num_trees_total)

  # collect all simulation files from the current directory
  # stats
  # Format for a stats file is:
  #id proba iteration burning_trees burned_trees burned_trees_total burned_trees_total_perc
  # run-id 0.35 1 1 4  0
  def stats_burning(self):
    cmd = "ls  %s/*-stats.txt" % self.options.odir
    files = os.popen(cmd).readlines()
    results = []
    proba_to_fraction = dict()
    for afile in files:

      lines = open(afile.rstrip()).readlines()
      if len(lines)==1: # header only found: nothing has been dumped
        continue

      # from line, retrieve run id + proba + fraction of burned trees
      last_line = lines[-1].rstrip()
      aux = re.split("\s+", last_line) # split the last line using blanks: we expect 5 numbers
      simul_id = int(aux[0])
      proba = float(aux[1])
      fraction = float(aux[-1])

      if proba in proba_to_fraction:
        proba_to_fraction[ proba ].append(fraction)
```

TIPE

```python
      if proba in proba_to_fraction:
        proba_to_fraction[ proba ].append(fraction)
      else:
        proba_to_fraction[ proba ] = [fraction]

    # for each proba value: pick the median fraction
    for proba in proba_to_fraction.keys():
      m = statistics.median( proba_to_fraction[proba] )
      results.append([proba, m])

    xs = [r[0] for r in results]
    ys = [r[1] for r in results]
    plt.plot(xs, ys,  '--bo')
    plt.xlabel('Probabilite')
    plt.ylabel('% arbres brules')

    dfname = "%s/%s-burned-trees-topo%s-trim%.2f-res.png" % \
      (self.options.odir, self.ofname_prefix, self.options.topo, self.options.trim_proba)
    plt.savefig(dfname)
    plt.clf()
    # plt.show()

    cmd = "cp %s ~/attach/tipe" % dfname
    os.system(cmd)

  def stats_p_hat(self):
    proba_to_fraction = dict()

    xs = []; ys = []; zs = []
    lines = open(self.dfname_phat).readlines()
    for line in lines:
      aux = re.split("\s+", line.rstrip())
      proba = float(aux[0])
      phat = float(aux[1])
      if proba in proba_to_fraction:
        proba_to_fraction[ proba ].append(phat)
      else:
        proba_to_fraction[ proba ] = [phat]

    tmp = []
    for proba in proba_to_fraction.keys():
      tmp.append( (proba, statistics.median(proba_to_fraction[proba])) )
    ord =  sorted(tmp)
    for t in ord:
      xs.append(t[0])
      ys.append(t[1])
      zs.append(t[0])

    # linear regression
```

```python
    # linear regression
    ls = []
    slope, intercept, r_value, p_value, std_err = stats.linregress(xs, ys)
    print("slope and r-squared:", slope, r_value**2)
    for x in xs:
      ls.append(intercept +slope*x)

    # plot: simulation
    plt.plot(xs, ys,  '--bo', color='r', label='Simuation')  # Simulation
    # plot: linear regression
    plt.plot(xs, ls, color='r',   label='Regression lineaire')
    # plot: y=x for reference
    plt.plot(xs, zs, color='g',   label='y=x')

    plt.xlabel('Proba')
    plt.ylabel('Estimation')

    plt.legend()
    dfname = "%s/%s-ML-estimation-topo%s-trim%.2f-res.png" % \
      (self.options.odir, self.ofname_prefix, self.options.topo, self.options.trim_proba)
    plt.savefig(dfname)
    plt.clf()
    # plt.show()

    cmd = "cp %s ~/attach/tipe" % dfname
    os.system(cmd)

  # return a pair (number of burned trees, new matrix)
  def one_iteration(self, iteration):
    Mpl = self.M_forest # M plus one: forest at the nex iteration
    n=len(self.M_forest); m=len(self.M_forest[0])

    burned_trees = 0
    burning_trees = 0

    # now compute for this iteration: #new_burning #candidate
    candidates = set()
    burning_new = set()

    for i in range(n):
        for j in range (m):
            if self.M_forest[i,j,0] == burning:
              neighbors_offsets = alive_neighbors_offsets_of_burning_voxel(self.M_forest, i, j, self.options.

              # tree is burning, but was not lit during this iteration
              if self.M_forest[i,j,1] != iteration:
                for x in neighbors_offsets:
                    a = x[0]; b = x[1]

                    candidates.add( (i+a, j+b) )
```

```python
                    candidates.add( (i+a, j+b) )

                    # fire transmission prob is the sum of the default proba + contribution of the wind;
                    # the latter is the dot product between offset vector and the wind vector
                    # if sucess: the neighbor will burn too
                    r = random.randint(1,100)
                    if r <= (self.options.p*100+ 10*(a*self.wind[0]+b*self.wind[1])):
                        burning_new.add( (i+a,j+b) )
                        Mpl[i+a,j+b]=[burning, iteration]
                        self.M_rgb[i+a,j+b]=rgb_burning
                        burning_trees += 1

              # tree burned during self.burning_time steps: dead
              if iteration-self.M_forest[i,j,1] >= self.options.burning_time:
                Mpl[i][j][0] = burned
                self.M_rgb[i][j] = rgb_burned
                burned_trees += 1

    # update the forest data structure for the  next iteration
    #msg = "iteration - burning trees - burned trees: %s %s %s" % (iteration, burning_trees, burned_trees)
    #print(msg)

    for i in range(n):
        for j in range (m):
            self.M_forest[i,j]=Mpl[i,j]

    # record the information to estimate p_hat
    if burning_trees != len(burning_new):
      sys.exit("problem")
    self.p_hat_estimation.append( [len(burning_new), len(candidates) ] )
    return (burning_trees, burned_trees, Mpl)


#n= taille du carré, p= probabilité de transmission, d-nombre de départs de feu, t=nombre de tours durant les
  def simulation(self):
    n=len(self.M_forest); m=len(self.M_forest[0])
    msg = "Running iteration on matrix of size %s and %s for %s voxels" % (n,m,n*m)
    print(msg)

    c = n//2; d = m//2
    self.M_forest[c,d] = [burning,0]
    self.M_forest[c+1, d] = [burning,0]
    self.M_forest[c, d+1] = [burning,0]
    self.M_forest[c+1, d+1] = [burning,0]

    iteration=1
    burned_trees_total = 0

    f=open(self.dfname_stats, "w") # open file to append a line
    f.write("#proba iteration burning trees burned trees burned trees total burned trees total perc\n")
```

```python
f=open(self.dfname_stats, "w") # open file to append a line
f.write("#proba iteration burning_trees burned_trees burned_trees_total burned_trees_total_perc\n")
f.close()

while True:
  (burning_trees, burned_trees, Mpl) = self.one_iteration(iteration)
  # print("Iter burning_trees, burned_trees", iteration, burning_trees, burned_trees)
  burned_trees_total += burned_trees

  burned_perc = 100.0*burned_trees_total / self.num_trees_total
  if burning_trees == 0:
    break

  # store imagel
  # output_filename = "forestM-windx%s-windy%s-iteration%s.png" % (self.wind[0], self.wind[1], iteration)
  # X = Image.fromarray(Mpl)
  # X.save(output_filename)

  if self.options.picts:
    X = Image.fromarray(self.M_rgb)
    self.dfname_pict       = "%.2f-iteration%s-pict.png" % (self.options.p, iteration)
    X.save(self.dfname_pict)

  # update statistics
  f=open(self.dfname_stats, "a") # open file to append a line
  line = "%d %.2f %s %s %s %2.f" % \
    (self.options.id, self.options.p, iteration, burning_trees, burned_trees, burned_perc)
  print("Simul-id / proba  / iteration / burning trees / burned trees:", line)
  f.write(line + "\n")
  f.close()

  iteration = iteration+1

# when done: store the last pict in any case
X = Image.fromarray(self.M_rgb)
X.save(self.dfname_pict_final)

# at least 2 iterations needed
if  iteration == 1:
  return

# estimation of the proba from the trees reached / starting to burn at each iteration
nn = 0; mm = 0
for k in range(0, len(self.p_hat_estimation)):
  nn += self.p_hat_estimation[k][0] # new burning
  mm += self.p_hat_estimation[k][1] # candidates
p_hat = float(nn)/mm
print("Estimation: ", p_hat)

f=open(self.dfname_phat, "a") # open file to append a line
```

```
        f=open(self.dfname_phat, "a") # open file to append a line
        line = "%.2f %.2f" % (self.options.p, p_hat)
        f.write(line + "\n")
        f.close()

    def run(self):
        (dd, ff) = os.path.split(self.options.fname)
        self.ofname_prefix      = re.sub("\.\w{3}","",ff);    # name of the file containing the image

        # output directory: exists, or create
        if self.options.odir:
          if not os.path.exists(self.options.odir):
            sys.exit("You passed a directory which does not exist")
        else:
          # prepare the prefix of output filenames
          self.options.odir = "%s-topo%s-windx%s-windy%s-trim%.2f" % \
            (self.ofname_prefix, self.options.topo,  self.wind[0], self.wind[1], self.options.trim_proba)

          # run mode only: if self.options.odir pre-exists, remove and re-create
          if self.options.mode == "run" and os.path.exists(self.options.odir):
            cmd = "rm -rf %s" % self.options.odir; print(cmd); os.system(cmd)
          cmd = "mkdir %s"  % self.options.odir; print(cmd); os.system(cmd)

        self.dfname_stats      = "%s/proba%.2f-id%s-stats.txt" % (self.options.odir, self.options.p,  self.option
        self.dfname_pict_final = "%s/proba%.2f-id%s-pict.png"  % (self.options.odir, self.options.p, self.options
        self.dfname_phat       = "%s/p-hat.txt" % self.options.odir


        # run the simulation
        if self.options.mode == "run":
          # directory into which results are stored
          self.M_rgb = get_rgb_matrix(self.options.fname)
          self.get_forest_matrix(self.M_rgb)
          self.simulation()
        # run the stats
        elif self.options.mode == "stats":
          self.stats_burning()
          self.stats_p_hat()
        else:
          print("Provide the option run for the simulation, and stats for the statistics")


#i#################################################################################
# Options of the simulation in the command line of the shell
parser = argparse.ArgumentParser(description='My parser')

parser.add_argument("-f", "--fname", dest="fname", default="data/foret-ciel-small.jpg", help="Image filename"
parser.add_argument("--trim", dest="trim_proba", type=float, default=0.0, help="Trim proba. If 0 (default), n

parser.add_argument("-p", "--proba", dest="p", default=0.3 type=float, help="Fire propagation probability")
```

TIPE

```python
      if self.options.mode == "run" and os.path.exists(self.options.odir):
        cmd = "rm -rf %s" % self.options.odir; print(cmd); os.system(cmd)
      cmd = "mkdir %s"  % self.options.odir; print(cmd); os.system(cmd)

    self.dfname_stats      = "%s/proba%.2f-id%s-stats.txt" % (self.options.odir, self.options.p,  self.option
    self.dfname_pict_final = "%s/proba%.2f-id%s-pict.png"  % (self.options.odir, self.options.p, self.options
    self.dfname_phat       = "%s/p-hat.txt" % self.options.odir


    # run the simulation
    if self.options.mode == "run":
      # directory into which results are stored
      self.M_rgb = get_rgb_matrix(self.options.fname)
      self.get_forest_matrix(self.M_rgb)
      self.simulation()
    # run the stats
    elif self.options.mode == "stats":
      self.stats_burning()
      self.stats_p_hat()
    else:
      print("Provide the option run for the simulation, and stats for the statistics")


#i###############################################################################
# Options of the simulation in the command line of the shell
parser = argparse.ArgumentParser(description='My parser')

parser.add_argument("-f", "--fname", dest="fname", default="data/foret-ciel-small.jpg", help="Image filename"
parser.add_argument("--trim", dest="trim_proba", type=float, default=0.0, help="Trim proba. If 0 (default), n

parser.add_argument("-p", "--proba", dest="p", default=0.3,type=float, help="Fire propagation probability")
parser.add_argument("-b", "--burning_time", dest="burning_time", default=3,type=int, help="Burning time")
parser.add_argument("-t", "--topo", dest="topo", default=4,type=int, help="Connectivity of a pixel: 4 or 8 ne

parser.add_argument("--odir", dest="odir", help="Output directory name for the results")
parser.add_argument("-i", "--id", dest="id", type=int, default=0, help="Simulation id when running repeats")
parser.add_argument("--pict", action="store_true", default=False, dest="picts", help="Store pictures at every

parser.add_argument("-m", dest="mode", help="Mode: run or stats")

options = parser.parse_args()

# The simulation
fire   = Fire_simulation(options) # create the instance of the simulation
fire.run()     # run the simulation

firesim.py
Displaying firesim.py.
```

TIPE

```python
import math
import numpy as np
from scipy.integrate import odeint
from random import *
import matplotlib.pyplot as plt

#constantes
taux_combustion=0.0001 #taux de combustion
p = 450 # KG/m3, masse volumique du sapin
h0 = 0.2 # tige de 10 centimetre de haut
r0 = h0/10
C_1= p*math.pi*(h0/10)**2*taux_combustion
T0 = 500 # k, température ambiante
Tig = 500 #temperature d'embrasement de la pelouse
Cp = 1500 #capacité thermique du sapin
s= 5.67 *10**(-8) # constante de steffan Boltzmann
e=0.05 #emissivite, comprise entre 0 et 1
Ti = 1500 #temperature de l'incendie et donc de la tige enflammee a t=
msol= 1 #  parallelepipede de sol de 1kg
csol = 0.00016 # capacite thermique de la pelouse
h=5 #coefficient de transfert thermique

################# COurbe 1

#masse du cylindre en fonction du temps
def hauteur_cylindre(t):
    if h0 - taux_combustion*t >= 0 :
        return (h0 - taux_combustion*t)
    else :
        return 0

def masse_cylindre (t) :
    m0 = 4*math.pi*r0**2*h0
    return (m0-C_1*t)
```

```python
def hauteur_cylindre(t):
    if h0 - taux_combustion*t >= 0 :
        return (h0 - taux_combustion*t)
    else :
        return 0

def masse_cylindre (t) :
    m0 = 4*math.pi*r0**2*h0
    return (m0-C_1*t)
    """
    if m0 - C_1*t >= 0 :
        return m0 - C_1*t
    else :
        return 0
    """
def surface_cylindre (t):
    return 2*math.pi*r0*(r0+hauteur_cylindre(t))

def model1 (T,t)  :
    dTdt = - (surface_cylindre(t)/(masse_cylindre(t)*Cp)) * (s*e*(T**4
    return dTdt

temps = np.linspace(0,20,20000) # tableaux des temps, 10 milles valeur
valeurs_T = odeint(model1, Ti, temps ) # obtention de la temperature c


#print (valeurs_T)
#print (len(valeurs_T))

def Temperature_tige(t):
    n=int(t/0.002)
    #print (valeurs_T[n][0])
    return valeurs_T[n][0]
```

```python

temps = np.linspace(0,20,20000) # tableaux des temps, 10 milles valeur
valeurs_T = odeint(model1, Ti, temps ) # obtention de la temperature c


#print (valeurs_T)
#print (len(valeurs_T))

def Temperature_tige(t):
    n=int(t/0.002)
    #print (valeurs_T[n][0])
    return valeurs_T[n][0]

Temperature_tige_liste = [Temperature_tige(t) for t in temps[:20000]]


def model2 (Ts,t):
    terme1=s*e*(Temperature_tige(t)**4-T0**4)
    terme2=h*(Temperature_tige(t)-T0)
    dTsdt = 0.25*(1/(msol*csol))*(surface_cylindre(t)/(masse_cylindre(
    return dTsdt


valeurs_Tsol = odeint(model2,T0,temps) #initalement le sol est a tempe

def Temperature_sol(t):
    n=int(t/0.002)
    #print (valeurs_T[n][0])
    return valeurs_Tsol[n][0]

Temperature_sol_liste = [Temperature_sol(t) for t in temps[:20000]]


################### Courbe 2
```

TIPE

```python


plt.plot (temps, Temperature_tige_liste, label='0.1m' )
plt.plot (temps, Temperature_tige_liste_2, label='0.5m')
plt.plot (temps, Temperature_tige_liste_3, label='1m')
plt.xlabel('Temps (s)')
plt.ylabel('Température Tige (K)')
plt.legend()
plt.show()


#print (Temperature_sol_liste, len(Temperature_sol_liste))

tableau_tig = y1 = [Tig for i in temps] # tableau de T=Tig cste
plt.plot (temps, Temperature_sol_liste_3, label='0.1m')
plt.plot (temps, Temperature_sol_liste_2, label='0.5m')
plt.plot (temps, Temperature_sol_liste, label='1m')
plt.plot(temps,tableau_tig, label="Température d'ignition")
plt.xlim([0, 18])
plt.ylim([0,1500 ])
plt.xlabel('Temps (s)')
plt.ylabel('Température Sol (K)')
plt.legend()
plt.show()
```