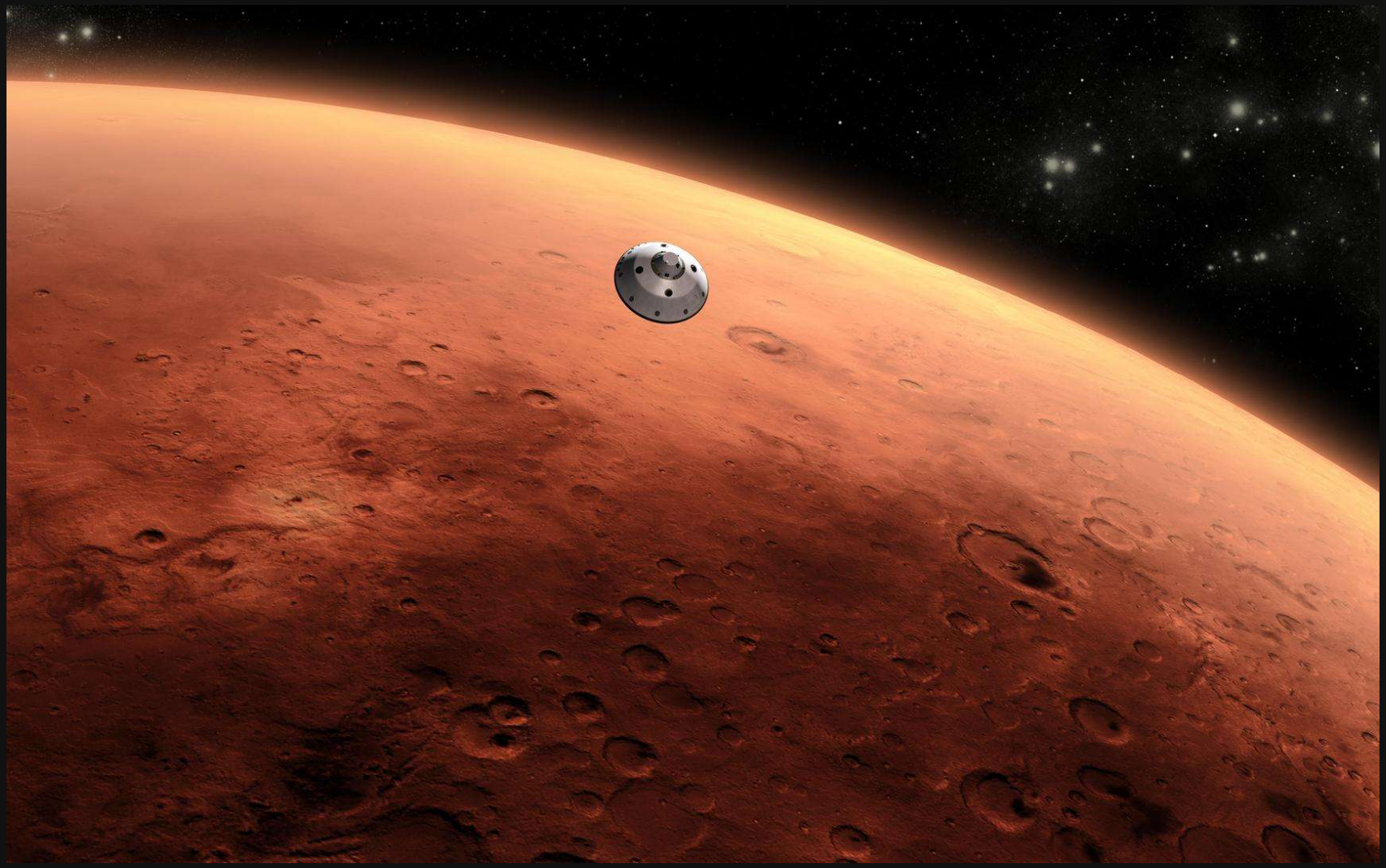
La colonisation de Mars

ARE - Dynamic

MOREAU Léonard
BOUSCARAT Tom
GUICHARD Victor
DZIRI Hakim

Problématique :

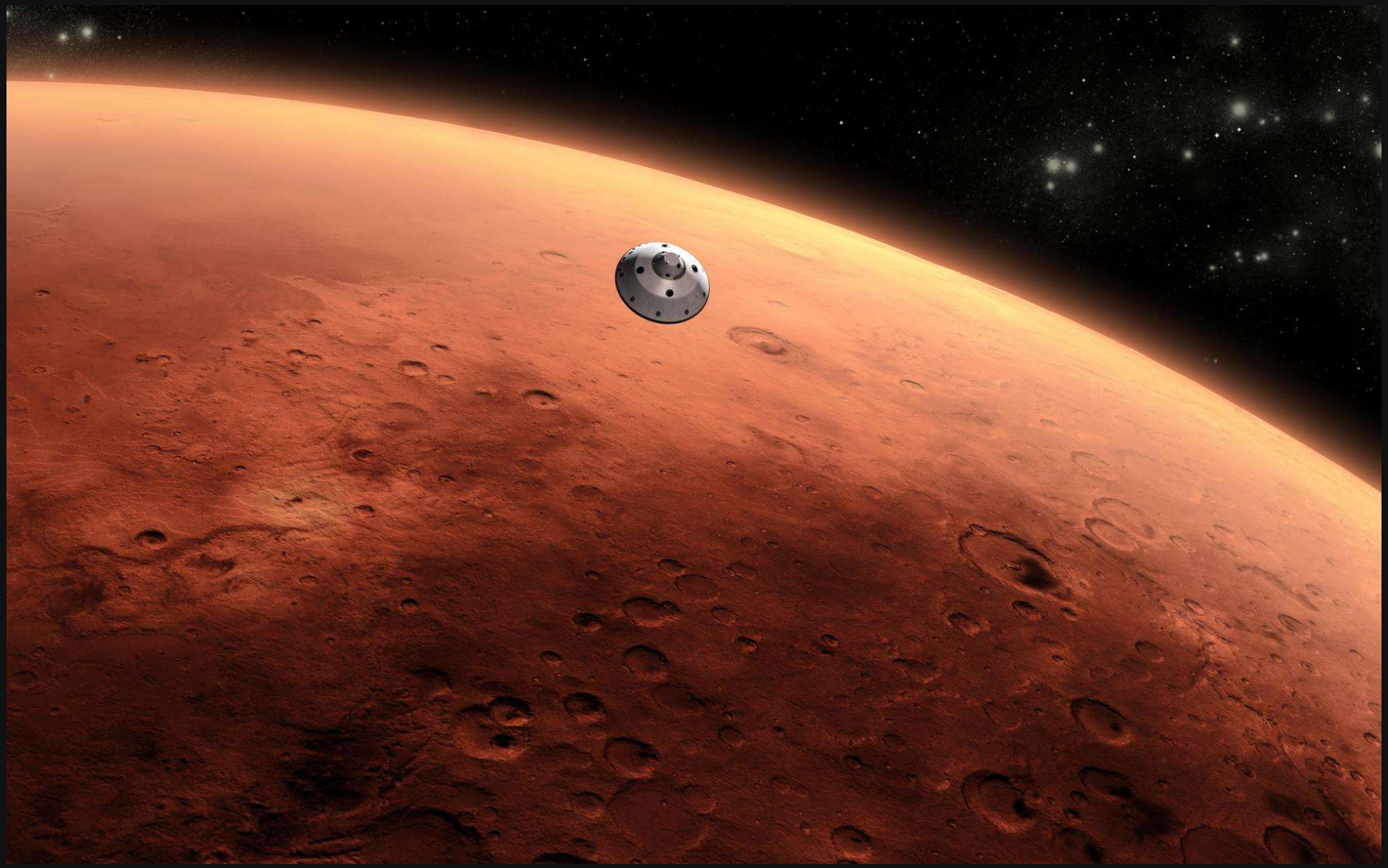
Comment assurer la survie d'une colonie sur Mars ?



Plan :

- Introduction
- Présentation d'un schéma d'évolution basique :
  - Naissances
  - Morts
- Différenciation des sexes
- Ressources
- Conclusion

Introduction



Présentation d'un schéma d'évolution basique :

En utilisant par exemple :

- Des dictionnaires de populations.
- Différents paramètres tels que des dictionnaires de mortalités et de naissances.
- Des fonctions mettant en jeu les différents dictionnaires initiaux pour renvoyer de nouvelles données (tel que les dictionnaires mis a jours, l'évolution de la population au fil des décennies,...).

Naissances

Réalisation d'un modèle basique de naissances en fonction d'une population (sans différenciation des genres).

In [129]:
```python
pop_init = {0 : 0,
    10 : 0,
    20 : 50,
    30 : 50,
    40 : 10,
    50 : 5,
    60 : 0,
    70 : 0,
    80 : 0,
    90 : 0}
```

Fonction qui donne le nombre total de naissances en fonction du dictionnaire de population

In [130]:
```python
def nb_naissances(p1, p2):
    '''
    dict[int:int] * dict[int:dict[int:float]] -> int
    renvoie le nombre de naissances pendant 10 ans
    '''
    a = 0
    nv_enfant = 0
    for a in p1:
        b = p1[a]
        i = 0
        while i < b:
            nv_enfant += number_of_descendants(p2, a)
            i += 1
    return nv_enfant

nb_naissances(pop_init,p)
```

Out[130]: 53

Morts

Quelques fonctions pour représenter les morts (toujours sans différenciation des genres).

In [133]:
```python
def plata_o_plomo(pmort, age):
    '''
    dict[int:float]*int->int
    renvoie 0 si vivant et 1 si mort
    '''
    u = np.random.random()
    if u < pmort[age]:
        return 1
    else :
        return 0
plata_o_plomo(pmort,30)
```

Out[133]: 0

Puis on créé une fonction qui va nous donner le nombre de mort pour une generation (decade)

In [134]:
```python
def mort_gen(pmort, nb_gen, age):
    '''
    dict[int:float]*int*int->int
    Renvoie le nombre de mort d'une generation
    '''
    i = 0
    morts = 0
    while i < nb_gen:
        morts += plata_o_plomo(pmort, age)
        i += 1
    return morts

mort_gen(pmort, 50, 30)
```

Out[134]: 1

In [131]:
```python
def pop_decade(pop,p):
    '''
    dict[int:int] * dict[int:dict[int:float]] -> dict[int:int]
    renvoie le dict de population apres une decade (lire avec l'accent svp)
    '''
    nv_enfant = nb_naissances(pop, p)
    pop_ev = {}
    i = 0
    while i < 90:
        pop_ev[(i + 10)] = pop[i]
        i += 10

    pop_ev[0] = nv_enfant

    return pop_ev


pop_decade(pop_init,p)

```

Out[131]: {0: 55, 10: 0, 20: 0, 30: 50, 40: 50, 50: 10, 60: 5, 70: 0, 80: 0, 90: 0}

On introduit ensuite à l'aide d'un dictionnaire les probabilités de mortalités.

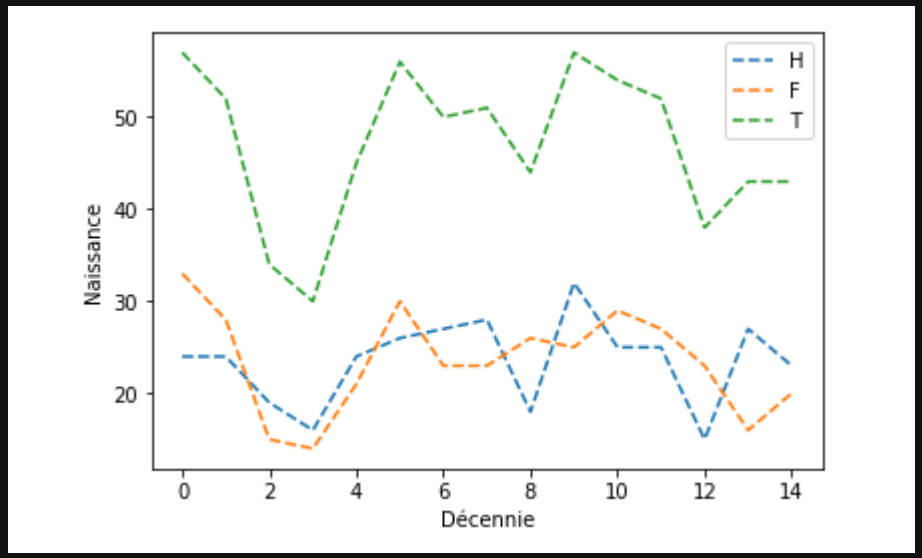
In [132]:
```python
# Dictionnaire représentant la distribution de probabilité de mourir.
# clef = age et valeur = probabilité

pmort = {0 : 0.1,
    10 : 0.02,
    20 : 0.02,
    30 : 0.05,
    40 : 0.08,
    50 : 0.09,
    60 : 0.12,
    70 : 0.15,
    80 : 0.30,
    90 : 1.0}
```

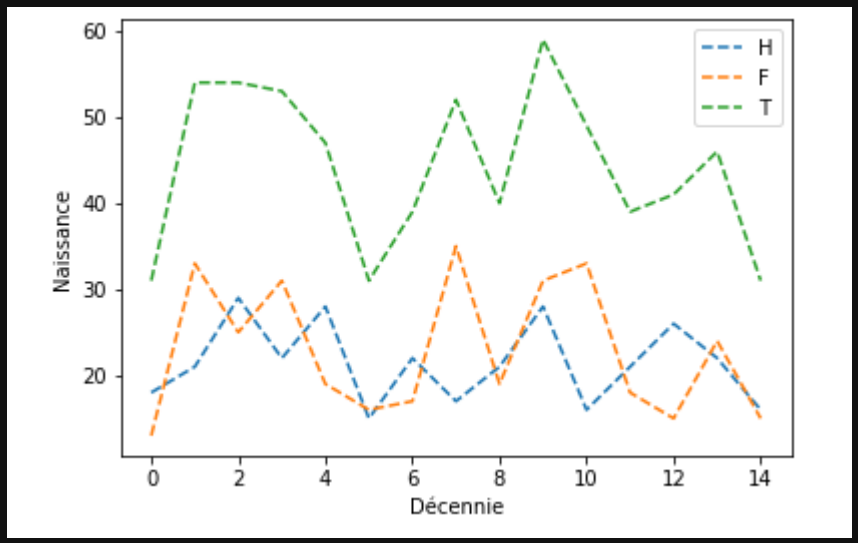
Différenciation des sexes

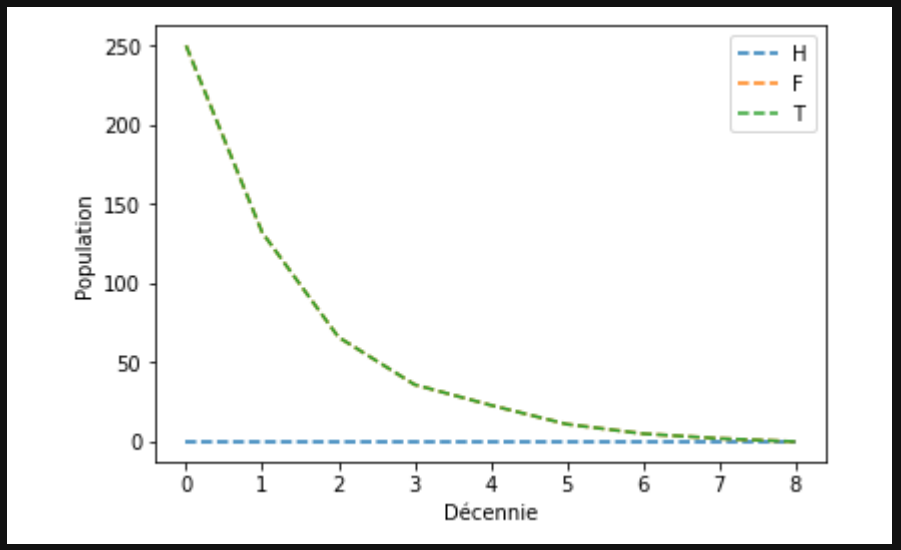
Nous avons imposé des conditions sur les fonctions de naissances et de morts pour rendre le modèle plus réaliste.

- On différencie la population en deux groupes représentant les hommes et les femmes, qui évoluent différemment dans le temps.
- Il faut nécessairement un homme et une femme pour engendrer une descendance.
- La population ne survie pas si l'un des deux sexe n'est pas présent.

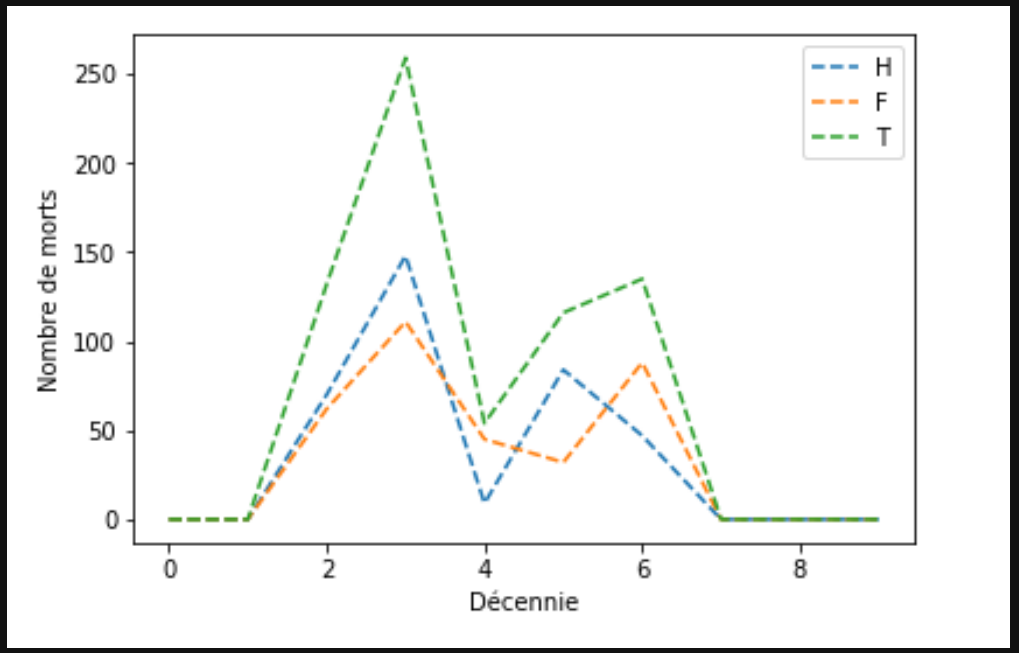
Naissances après différenciation des sexes







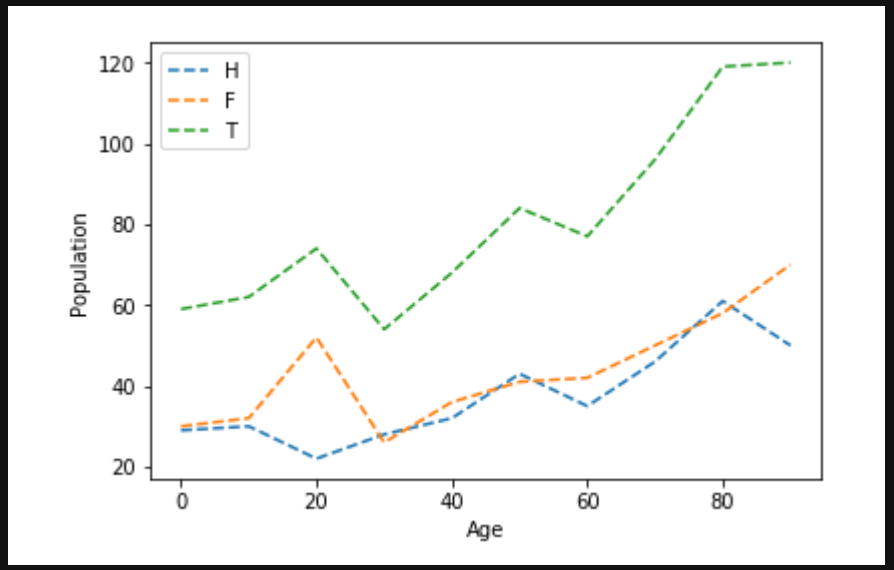
Morts après différenciation des sexes

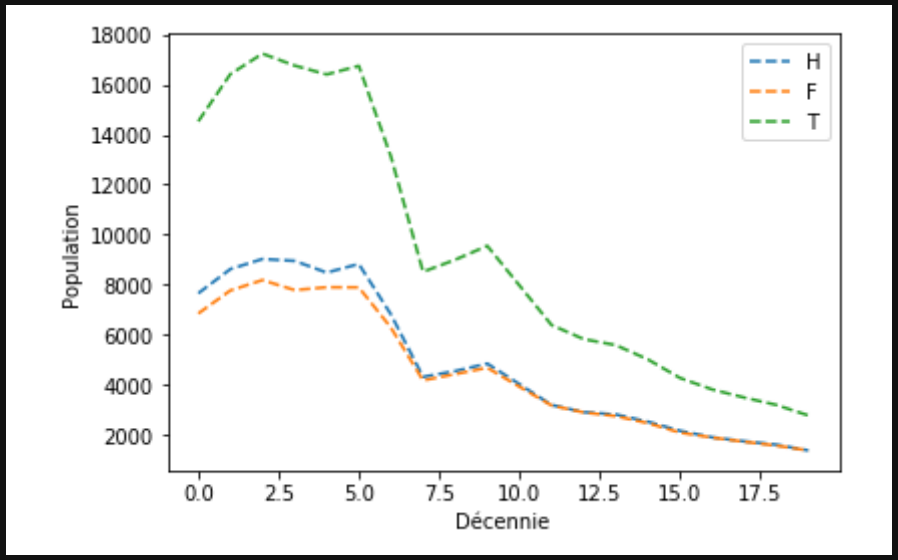


Ressources

Impact sur le dictionnaire de mortalité

Evolution de la population au cours du temps





Conclusion

Présentation d'un schéma d'évolution basique :

In [135]:
```python
def pop_vivante(pmort, pop):
    '''
    dict[int:float]*dict[int:int]->dict[int:int]
    Renvoie la population moins les morts
    '''

    pop_ev_ok = {}
    i = 0
    for i in pop :
        pop_ev_ok[i] = pop[i] - mort_gen(pmort, pop[i], i)
    return pop_ev_ok

pop_vivante(pmort, pop_init)
```

Out[135]: {0: 0, 10: 0, 20: 49, 30: 47, 40: 9, 50: 4, 60: 0, 70: 0, 80: 0, 90: 0}

On écrit ensuite la fonction finale qui nous donne après 10 ans la population (naissance puis mort).

In [136]:
```python
def pop_next(pop, p, pmort):
    """
    dict[int:int] * dict[int:dict[int:float]] * dict[int:float] -> dict[int:int]
    renvoie la population a la génération suivante
    """
    popnext = pop_decade(pop, p)
    popnext = pop_vivante(pmort, popnext)

    return popnext

print("Population à la génération suivante")
pop_next(pop_init, p, pmort)
```

Population à la génération suivante

Out[136]: {0: 53, 10: 0, 20: 0, 30: 50, 40: 43, 50: 10, 60: 4, 70: 0, 80: 0, 90: 0}

In [133]:
```python
def plata_o_plomo(pmort, age):
    '''
    dict[int:float]*int->int
    renvoie 0 si vivant et 1 si mort
    '''
    u = np.random.random()
    if u < pmort[age]:
        return 1
    else :
        return 0
plata_o_plomo(pmort,30)
```

Out[133]: 0

Puis on créé une fonction qui va nous donner le nombre de mort pour une generation (decade)

In [134]:
```python
def mort_gen(pmort, nb_gen, age):
    '''
    dict[int:float]*int*int->int
    Renvoie le nombre de mort d'une generation
    '''
    i = 0
    morts = 0
    while i < nb_gen:
        morts += plata_o_plomo(pmort, age)
        i += 1
    return morts

mort_gen(pmort, 50, 30)
```

Out[134]: 1

In [131]:
```python
def pop_decade(pop,p):
    '''
    dict[int:int] * dict[int:dict[int:float]] -> dict[int:int]
    renvoie le dict de population apres une decade (lire avec l'accent svp)
    '''
    nv_enfant = nb_naissances(pop, p)
    pop_ev = {}
    i = 0
    while i < 90:
        pop_ev[(i + 10)] = pop[i]
        i += 10

    pop_ev[0] = nv_enfant

    return pop_ev


pop_decade(pop_init,p)

```

Out[131]: {0: 55, 10: 0, 20: 0, 30: 50, 40: 50, 50: 10, 60: 5, 70: 0, 80: 0, 90: 0}

On introduit ensuite à l'aide d'un dictionnaire les probabilités de mortalités.

In [132]:
```python
# Dictionnaire représentant la distribution de probabilité de mourir.
# clef = age et valeur = probabilité

pmort = {0 : 0.1,
    10 : 0.02,
    20 : 0.02,
    30 : 0.05,
    40 : 0.08,
    50 : 0.09,
    60 : 0.12,
    70 : 0.15,
    80 : 0.30,
    90 : 1.0}
```

In [129]:
```python
pop_init = {0 : 0,
    10 : 0,
    20 : 50,
    30 : 50,
    40 : 10,
    50 : 5,
    60 : 0,
    70 : 0,
    80 : 0,
    90 : 0}
```

Fonction qui donne le nombre total de naissances en fonction du dictionnaire de population

In [130]:
```python
def nb_naissances(p1, p2):
    '''
    dict[int:int] * dict[int:dict[int:float]] -> int
    renvoie le nombre de naissances pendant 10 ans
    '''
    a = 0
    nv_enfant = 0
    for a in p1:
        b = p1[a]
        i = 0
        while i < b:
            nv_enfant += number_of_descendants(p2, a)
            i += 1
    return nv_enfant

nb_naissances(pop_init,p)
```

Out[130]: 53