

## Mars Lander

Comment décider d'une trajectoire permettant un atterrissage réussi sur mars?

Victor Angot - 755

CPGE Masséna - Nice

2020-2021

# Sommaire

1 Introduction

2 Premières approches

3 Algorithme génétique

4 Conclusion

5 Annexe

# Sommaire

## 1 Introduction

- Motivation et contexte du problème
- Modèle physique
- Définitions et notations

# Contexte

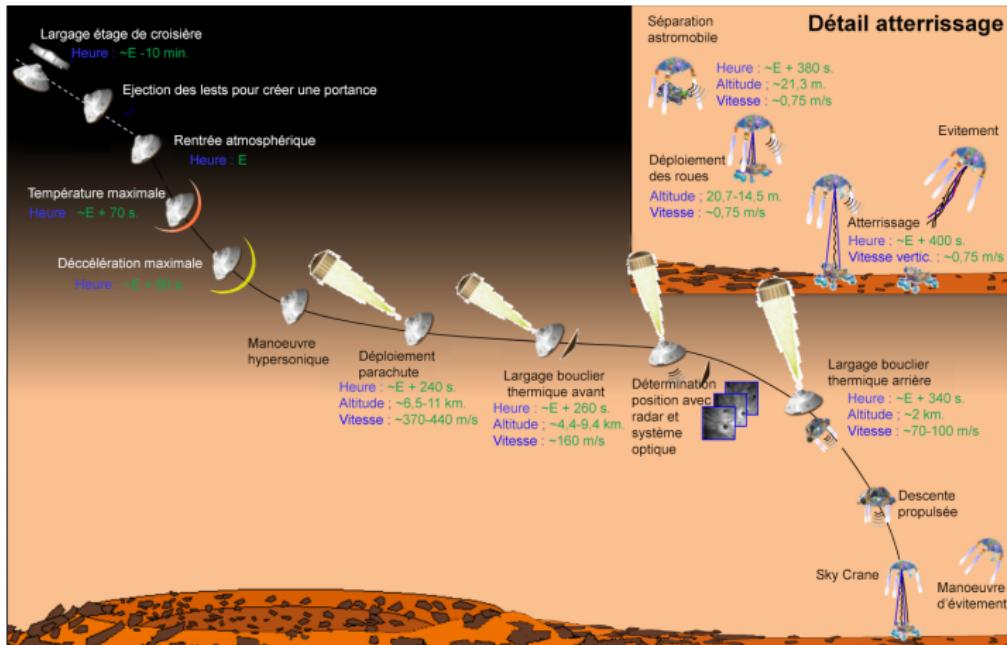


Figure – Déroulement de la descente de la sonde spatiale vers Mars (Wikipédia)

- Si l'engin spatial se dirige vers un site jugé dangereux, il peut utiliser la propulsion pour venir se poser sur un site adéquat.



Figure – Descente propulsée  
(Wikipédia)

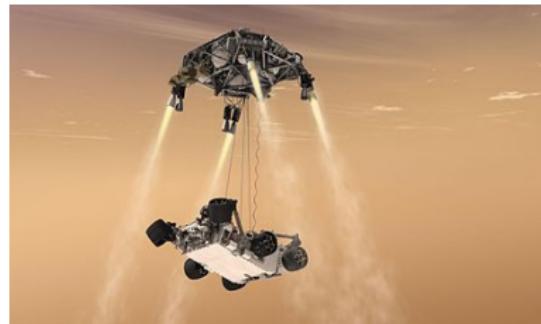


Figure – Etage de descente  
(Wikipédia)

# Sommaire

## 1 Introduction

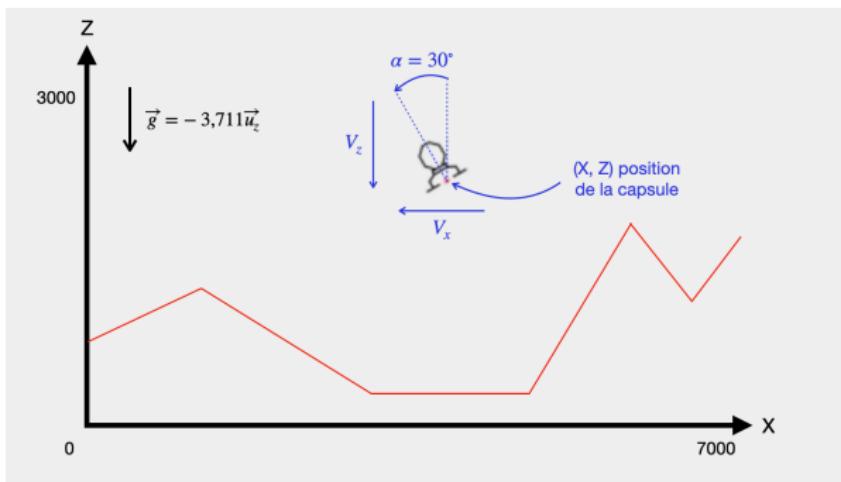
- Motivation et contexte du problème
- **Modèle physique**
- Définitions et notations

# Modèle physique

## Definition

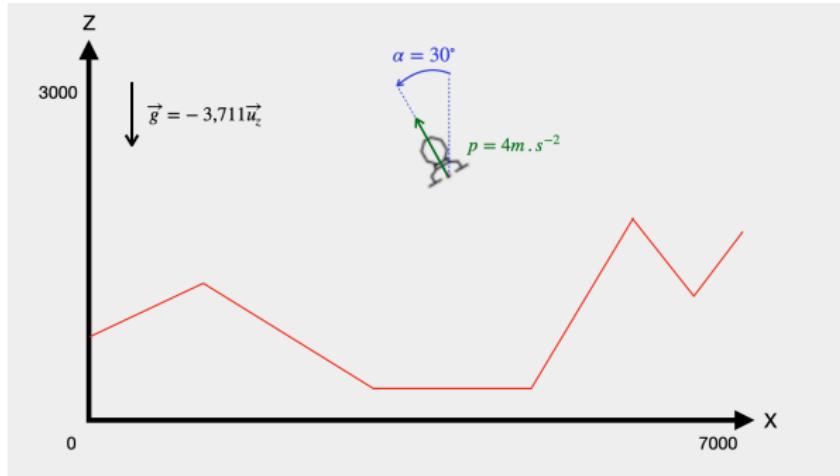
La capsule est caractérisé par le 7-uplet :

$$C = (X, Z, V_x, V_z, f, \alpha, p) \in (\mathcal{F}(\mathbb{R}, \mathbb{R})^N)^7$$



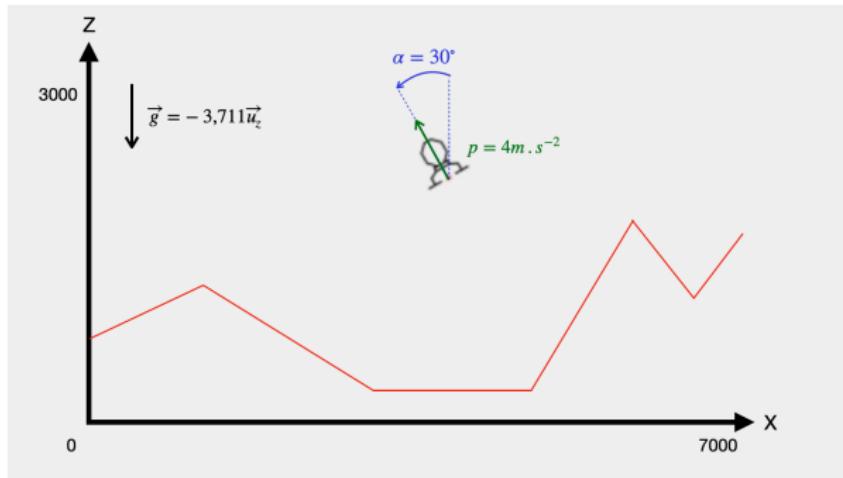
## Contraintes physiques du véhicule

- L'angle de rotation  $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  peut varier au maximum de  $\pm 15^\circ$  toutes les secondes.
- La poussée  $p \in [0, 4]$  peut varier de  $\pm 1 m.s^{-2}$  toutes les secondes.
- Générer une poussée de  $X m.s^{-2}$  coûte  $X$  litres de carburant.



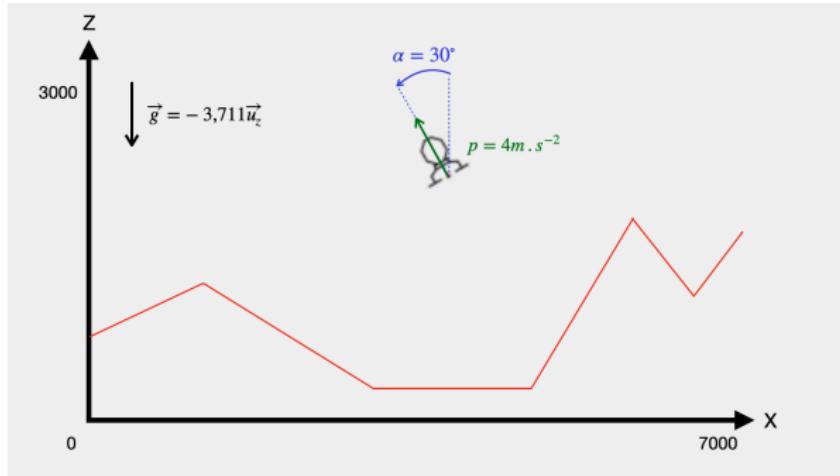
## Contraintes physiques du véhicule

- L'angle de rotation  $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  peut varier au maximum de  $\pm 15^\circ$  toutes les secondes.
- La poussée  $p \in [0, 4]$  peut varier de  $\pm 1 m.s^{-2}$  toutes les secondes.
- Générer une poussée de  $X m.s^{-2}$  coûte  $X$  litres de carburant.



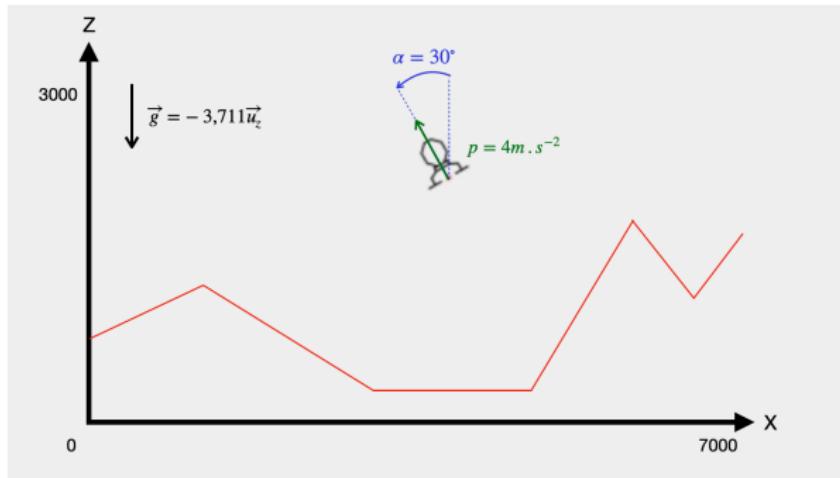
## Contraintes physiques du véhicule

- L'angle de rotation  $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  peut varier au maximum de  $\pm 15^\circ$  toutes les secondes.
- La poussée  $p \in [0, 4]$  peut varier de  $\pm 1 m.s^{-2}$  toutes les secondes.
- Générer une poussée de  $X m.s^{-2}$  coûte  $X$  litres de carburant.



## Contraintes physiques du véhicule

- L'angle de rotation  $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$  peut varier au maximum de  $\pm 15^\circ$  toutes les secondes.
- La poussée  $p \in [0, 4]$  peut varier de  $\pm 1 m.s^{-2}$  toutes les secondes.
- Générer une poussée de  $X m.s^{-2}$  coûte  $X$  litres de carburant.



## Modèle

On modélise la descente par une chute libre sans atmosphère.  
La capsule est ramené à son centre d'inertie.

## Equations

$$\ddot{X} = p \sin(\alpha) \Rightarrow X(t) = -\frac{1}{2}p \sin(\alpha)t^2 - V_{x0}t + X_0$$

$$\ddot{Z} = p \cos(\alpha) - g \Rightarrow Z(t) = \frac{1}{2}(p \cos(\alpha) - g)t^2 + V_{z0}t + Z_0$$

## Modèle

On modélise la descente par une chute libre sans atmosphère.  
La capsule est ramené à son centre d'inertie.

## Equations

$$\ddot{X} = p \sin(\alpha) \Rightarrow X(t) = -\frac{1}{2}p \sin(\alpha)t^2 - V_{x0}t + X_0$$

$$\ddot{Z} = p \cos(\alpha) - g \Rightarrow Z(t) = \frac{1}{2}(p \cos(\alpha) - g)t^2 + V_{z0}t + Z_0$$

# Sommaire

## 1 Introduction

- Motivation et contexte du problème
- Modèle physique
- Définitions et notations

# Définitions et notations

## Zone d'étude

Il existe une unique zone d'atterrissement.

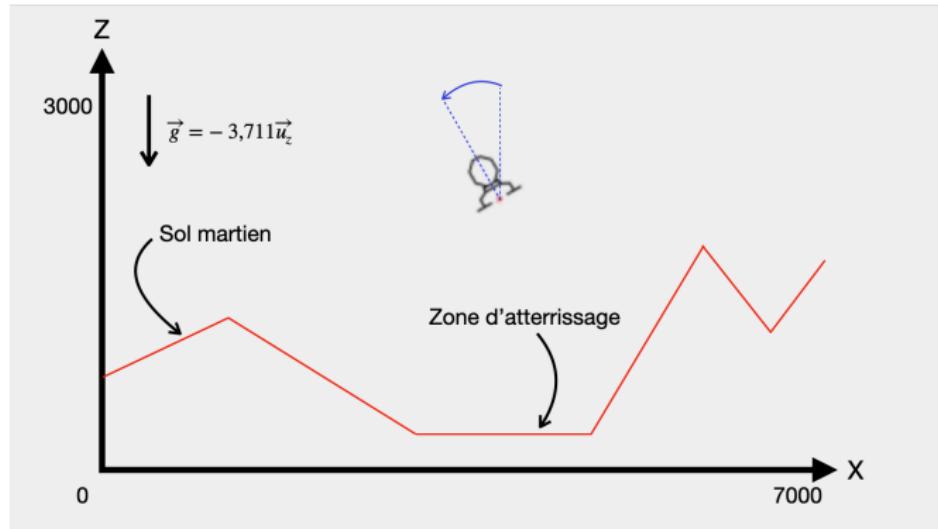
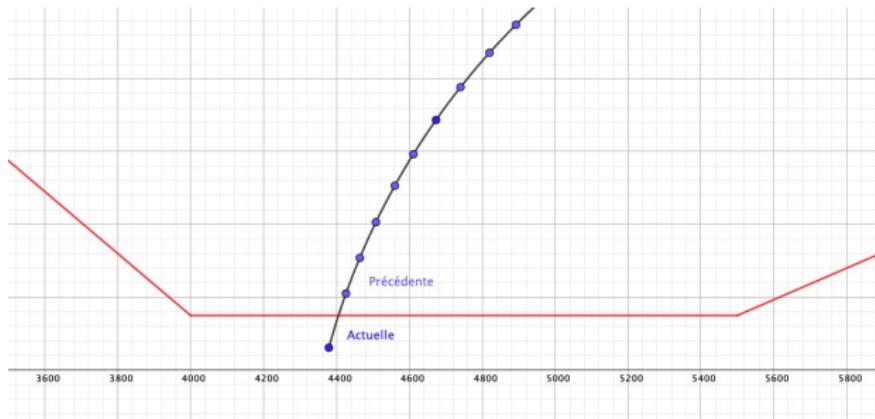


Figure – Paramétrage cartésien du plan d'origine  $O$

## Definition

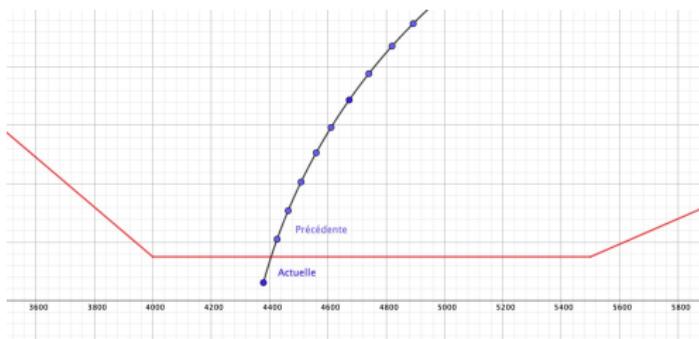
L'atterrissement est l'étape  $k \in \mathbb{N}$  tel que  $Z_{k-1} \leq h \leq Z_k$  où  $h \in \mathbb{R}$  est l'altitude du sol.



## Definition

Un atterrissage d'indice k est dit *réussi* lorsque on a :

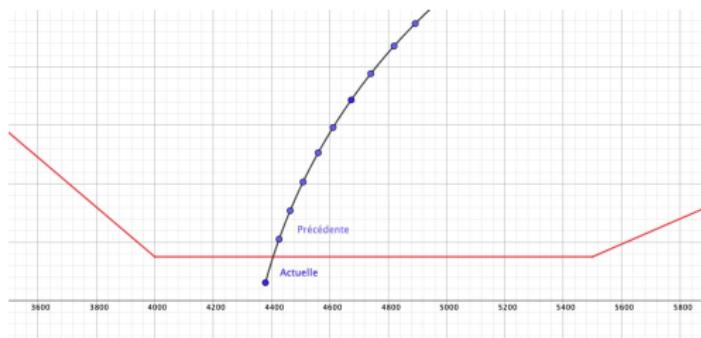
- Atterrissage vertical sur un sol plat ( $\alpha_k(t_f) = 0^\circ$ )
- Vitesse faible ( $Vx_k \leq 2m.s^{-1}$  et  $Vz_k \leq 4m.s^{-1}$ )



## Definition

Un atterrissage d'indice k est dit *réussi* lorsque on a :

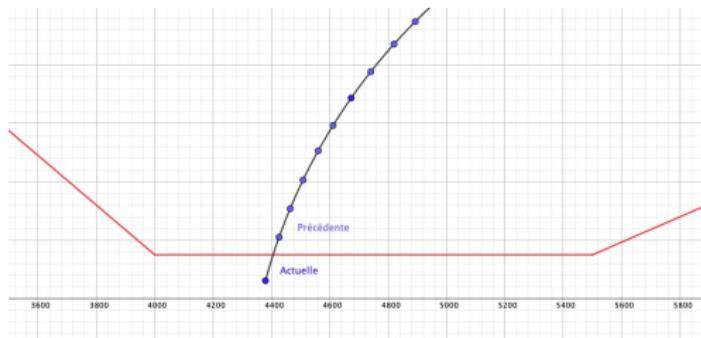
- Atterrissage vertical sur un sol plat ( $\alpha_k(t_f) = 0^\circ$ )
- Vitesse faible ( $Vx_k \leq 2m.s^{-1}$  et  $Vz_k \leq 4m.s^{-1}$ )



## Definition

Un atterrissage d'indice k est dit *réussi* lorsque on a :

- Atterrissage vertical sur un sol plat ( $\alpha_k(t_f) = 0^\circ$ )
- Vitesse faible ( $Vx_k \leq 2m.s^{-1}$  et  $Vz_k \leq 4m.s^{-1}$ )



## Definition

Une trajectoire est un  $(n+1)$ -uplet  $((x_0, z_0), \dots, (x_n, z_n)) \in (\mathbb{R}^2)^{n+1}$  avec  $n$  le nombre d'étapes avant l'atterrissement.



Figure – Exemple de trajectoire

# Sommaire

- 1 Introduction
- 2 Premières approches
- 3 Algorithme génétique
- 4 Conclusion
- 5 Annexe

# Sommaire

## 2 Premières approches

- Force brute
- Planification de trajectoire

# Force brute

## Dénombrément

Soit  $N$  le nombre de trajectoire possible, et  $n_0$  le nombre d'étapes minimale pour atterrir :

$$(2 \times 31)^{n_0} \leq N$$

# Sommaire

## 2 Premières approches

- Force brute
- Planification de trajectoire

# Sommaire

- 1 Introduction
- 2 Premières approches
- 3 Algorithme génétique
- 4 Conclusion
- 5 Annexe

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation
- Sélection
- Croisement
- Mutation
- Améliorations

# Principe de l'algorithme

## Origine

Modélisation informatique de la théorie de l'évolution de Darwin

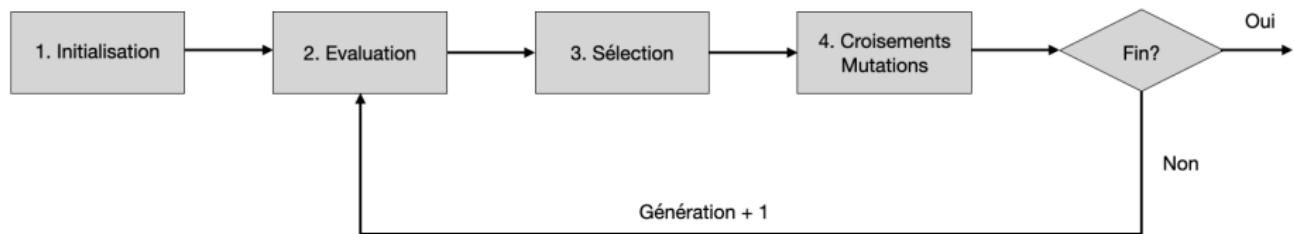


Figure – Principe de fonctionnement d'un algorithme génétique

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- **Initialisation**
- Evaluation
- Sélection
- Croisement
- Mutation
- Améliorations

# Initialisation

## Structure de donnée

- **Individu** : Instance de la structure de donnée.

$I = ((p_0, \alpha_0), \dots, (p_{n-1}, \alpha_{n-1})) \in (\mathbb{R}^2)^n$ . On fixe  $n=40$  .

## Implémentation python :

$$I = [[p_0, \alpha_0], [p_1, \alpha_1], \dots, [p_{n-1}, \alpha_{n-1}]]$$

# Initialisation

## Structure de donnée

- **Individu** : Instance de la structure de donnée.

$I = ((p_0, \alpha_0), \dots, (p_{n-1}, \alpha_{n-1})) \in (\mathbb{R}^2)^n$ . On fixe  $n=40$  .

## Implémentation python :

$$I = [[p_0, \alpha_0], [p_1, \alpha_1], \dots, [p_{n-1}, \alpha_{n-1}]]$$

## Fonction trajectoire

On dispose d'une application  $\phi : (\mathbb{R}^2)^{n+1} \rightarrow (\mathbb{R}^2)^{n+1}$  (codée par la fonction Trajectoire) qui associe un individu à sa trajectoire.

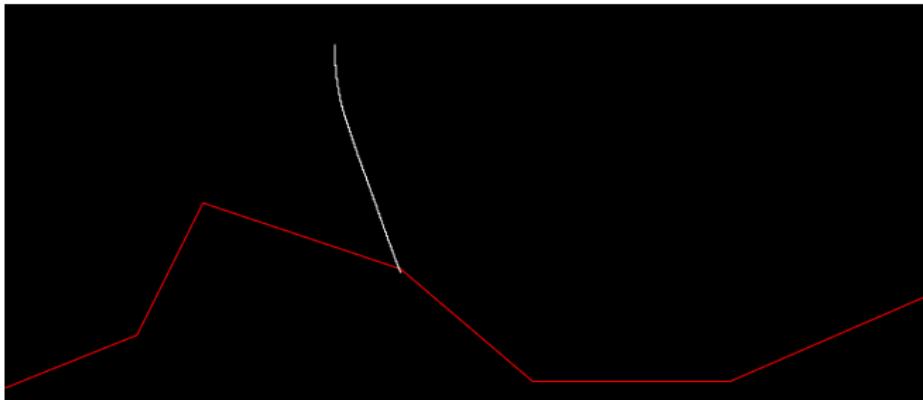


Figure – Représentation graphique d'un individu

## Population

La **population** est une collection d'individus. On fixe la taille à  $m=80$  individus pour assurer une grande diversité génétique.

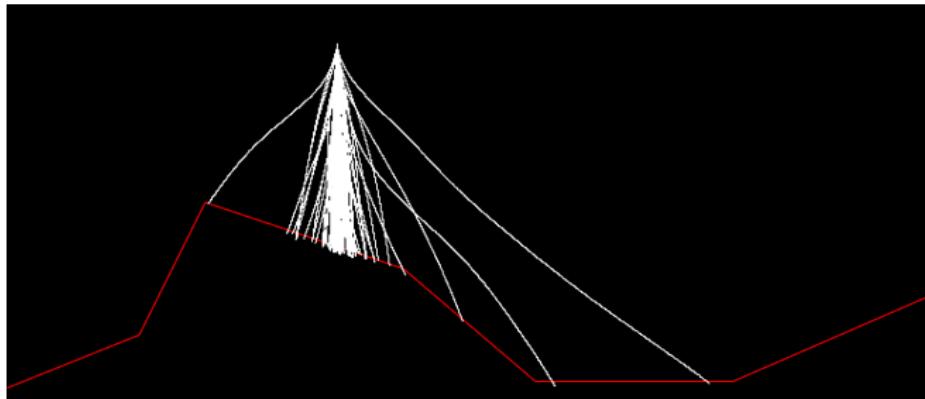


Figure – Représentation graphique d'une population (génération 0)

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation**
- Sélection
- Croisement
- Mutation
- Améliorations

# Evaluation

## Fonction d'évaluation

Fonction multi-critère dépendant de :

- ① la distance collision-zone d'atterrissage
- ② la vitesse de collision et à l'angle final

On cherchera ensuite à maximiser cette note.

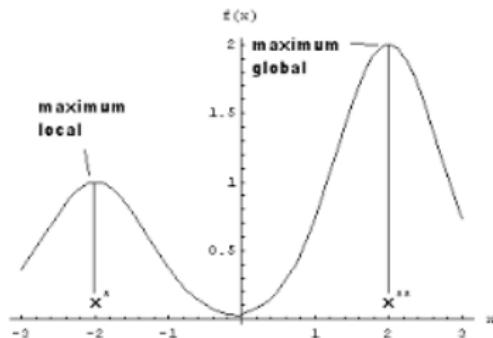


Figure – Répartition possible des notes (Google image)

# Evaluation

## Fonction d'évaluation

Fonction multi-critère dépendant de :

- ① la distance collision-zone d'atterrissage
- ② la vitesse de collision et à l'angle final

On cherchera ensuite à maximiser cette note.

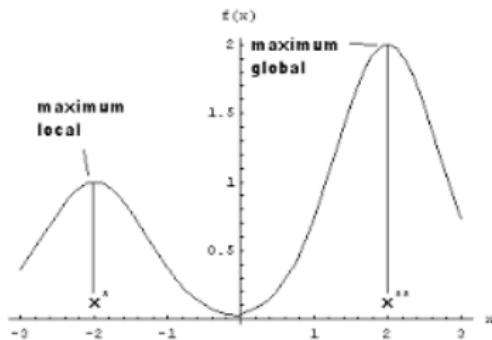


Figure – Répartition possible des notes (Google image)

# Evaluation

## Fonction d'évaluation

Fonction multi-critère dépendant de :

- ① la distance collision-zone d'atterrissage
- ② la vitesse de collision et à l'angle final

On cherchera ensuite à maximiser cette note.

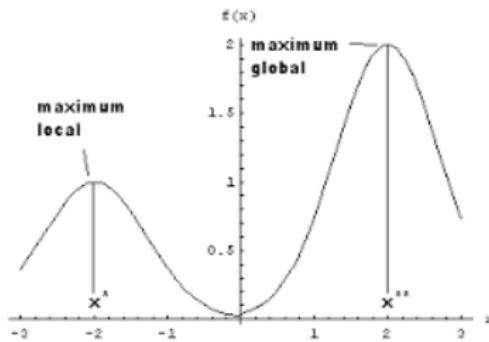


Figure – Répartition possible des notes (Google image)

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation
- Sélection
- Croisement
- Mutation
- Améliorations

# Sélection

## Méthode 1 : Roulette wheel

On associe à chaque individu un segment de longueur  $\propto$  fitness. On concatène ces segments sur un axe que l'on normalise entre 0 et 1.

Individu	Fitness
A	10
B	2.5
C	1
D	5

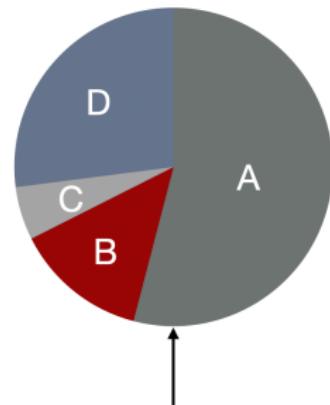


Figure – Exemple de roulette wheel

## Méthode 2 : Sélection élitiste

On trie les individus selon leur note et on conserve 20% de la population.

Amélioration : on conserve des individus avec une note faible. La probabilité choisie est de 0,05.

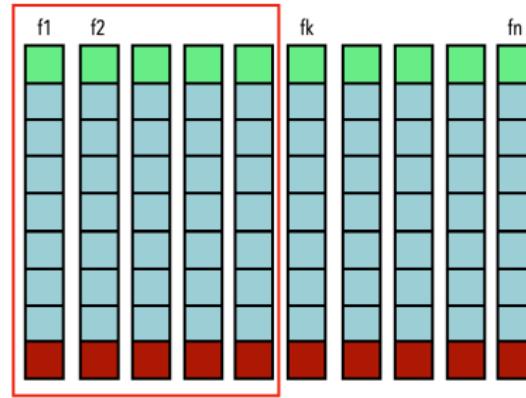
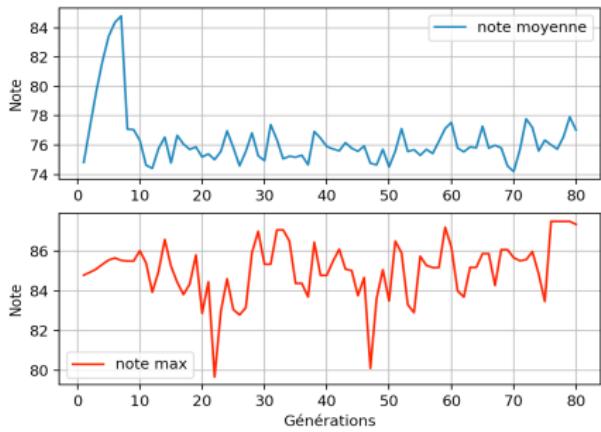
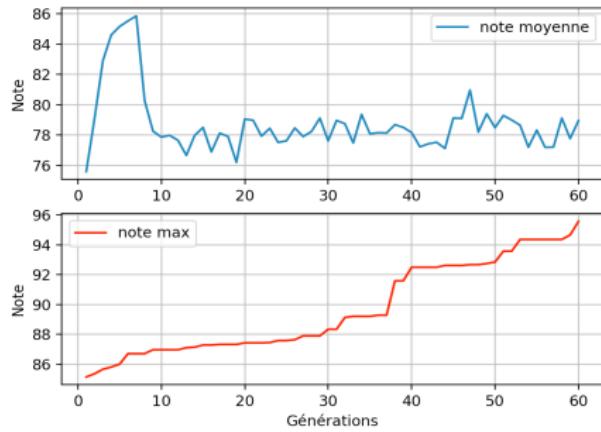


Figure – Sélection par élitisme ( $f_1 > \dots > f_n$ )



## Figure – Sans élitisme



## Figure – Avec élitisme

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation
- Sélection
- **Croisement**
- Mutation
- Améliorations

# Croisement

## Opérateur de croisement

On tire aléatoirement une position dans chacun des parents. On échange ensuite les deux sous-chaînes de chacun des deux chromosomes.

## Difficulté

L'individu croisé ou muté doit respecter les contraintes évoquées au slide n°8.

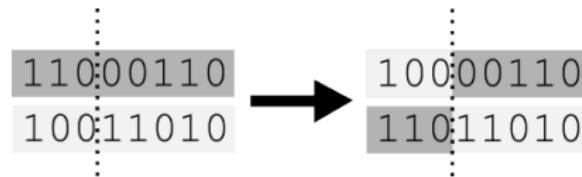


Figure – Croisement discret (Google image)

# Croisement

## Opérateur de croisement

On tire aléatoirement une position dans chacun des parents. On échange ensuite les deux sous-chaînes de chacun des deux chromosomes.

## Difficulté

L'individu croisé ou muté doit respecter les contraintes évoquées au slide n°8.

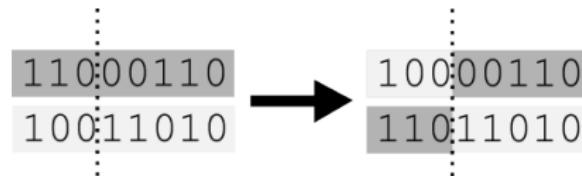


Figure – Croisement discret (Google image)

## Méthode 1 : Croisement discret (slicing crossover)

On croise deux individus en assurant une cohérence. On parcourt les deux parents pour trouver les points de croisements possibles.

Amélioration : dénoter tous les points de croisements possibles et effectuer des croisements sur un nombre aléatoire de ces points.

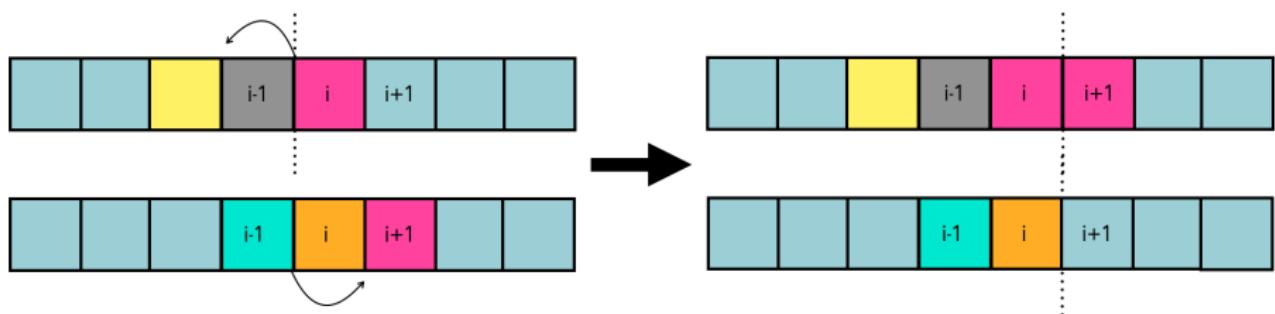


Figure – Croisement discret adapté

## Méthode 2 : Croisement continu ou "barycentrique"

Soit  $\lambda \in [0, 1]$ , soit  $P_1$  et  $P_2$  deux individus et  $P_1(i)$  et  $P_2(i)$  leur i-ème gènes. On pose  $C_1(i)$  et  $C_2(i)$  tel que :

$$\begin{cases} C_1(i) = \lambda P_1(i) + (1 - \lambda) P_2(i) \\ C_2(i) = (1 - \lambda) P_1(i) + \lambda P_2(i) \end{cases}$$

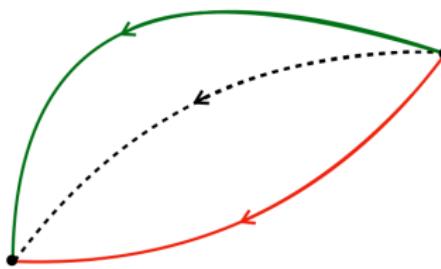


Figure – Fils engendré par deux parents

# Comparaison de ces deux méthodes (100 échantillons)

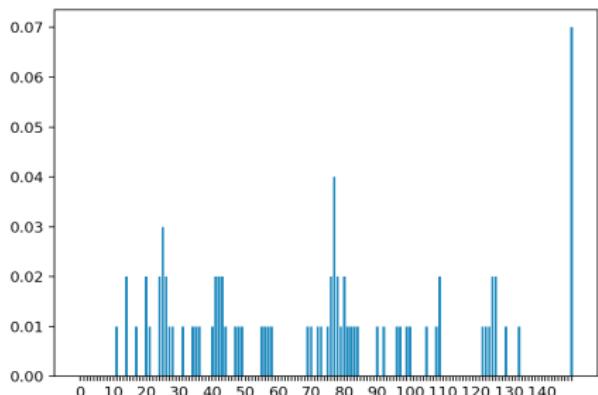


Figure – Croisement discret

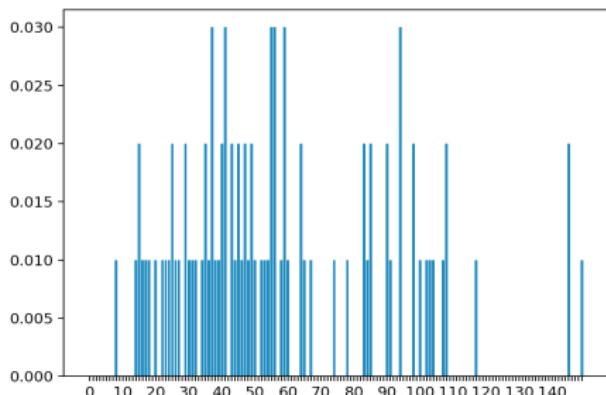


Figure – Croisement continu

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation
- Sélection
- Croisement
- Mutation**
- Améliorations

# Mutation

## Opérateur de mutation

L'opérateur de mutation apporte aux algorithmes génétiques la propriété d'ergodicité de parcours d'espace (rigoureusement la quasi ergodicité).

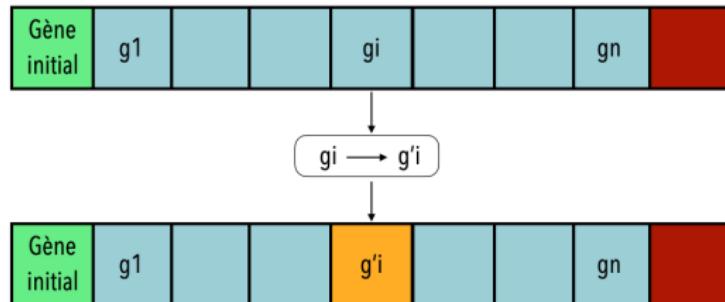


Figure – Domaine de mutation d'un individu

# Convergence en terme de générations (100 échantillons)

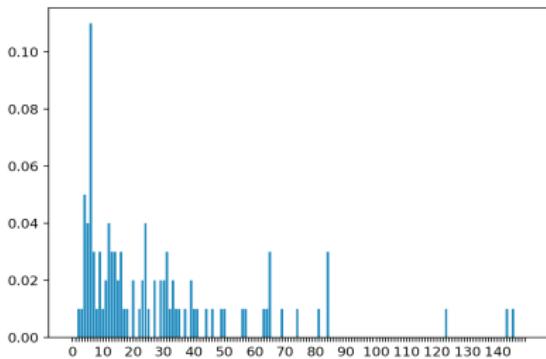


Figure – Sans mutation

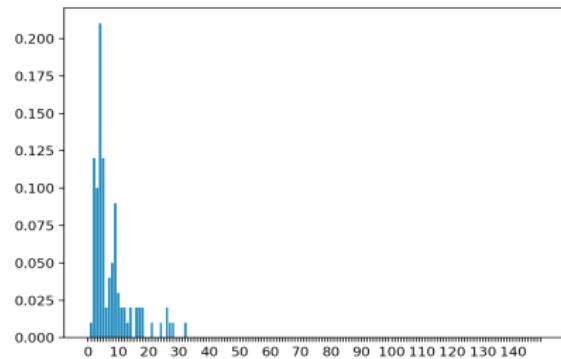


Figure – Avec mutation

# Sommaire

## 3 Algorithme génétique

- Principe de l'algorithme
- Initialisation
- Evaluation
- Sélection
- Croisement
- Mutation
- Améliorations

# Améliorations

## Réduction de l'espace de recherche

Soit  $I$  un individu dont l'atterrissement est d'indice  $k \in \mathbb{N}$ , on fait systématiquement muter  $\alpha_k$  tel que  $\alpha_k = 0$  afin de réduire l'espace de recherche (CSP).

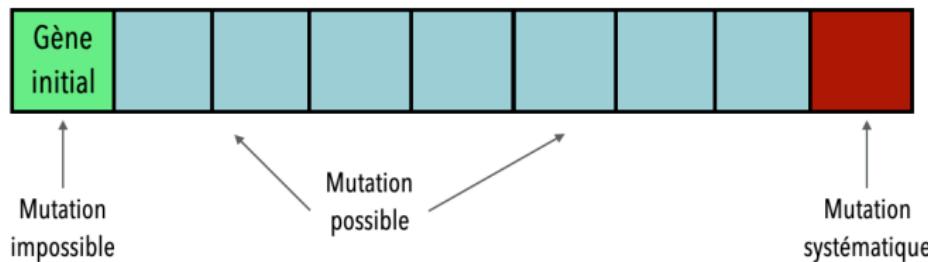


Figure – Domaine de mutation d'un individu

## Conservation de la diversité génétique

Si la note moyenne varie de moins de 1% en 10 générations on conserve les meilleurs individus et on en injecte de nouveaux.

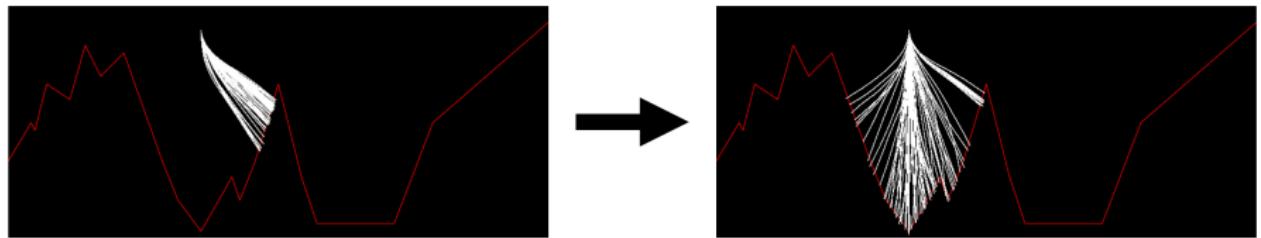


Figure – Exemple de convergence prématuée

## Zone de danger

### Zone interdite permettant :

- d'éviter les cas limites dangereux pour la capsule.
- de négliger l'incertitude sur l'arrêt de la trajectoire (utilisation de flottants).

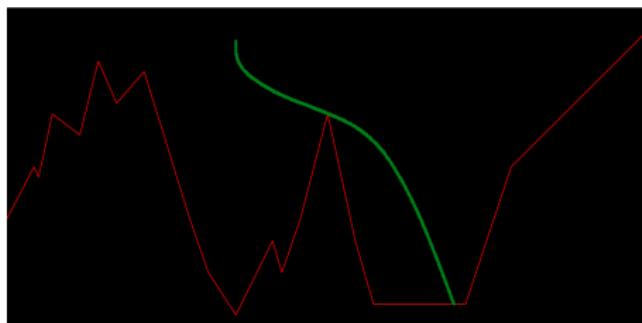


Figure – Sans zone de danger

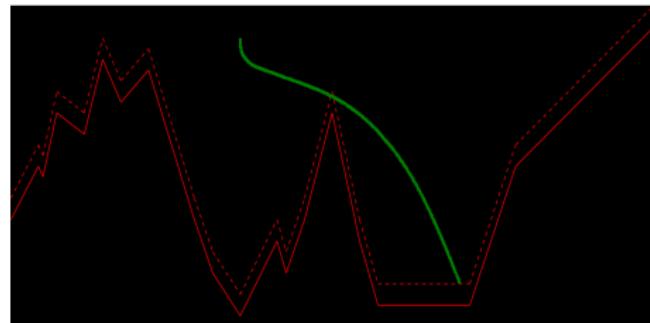


Figure – Avec zone de danger

## Zone de danger

Zone interdite permettant :

- d'éviter les cas limites dangereux pour la capsule.
- de négliger l'incertitude sur l'arrêt de la trajectoire (utilisation de flottants).

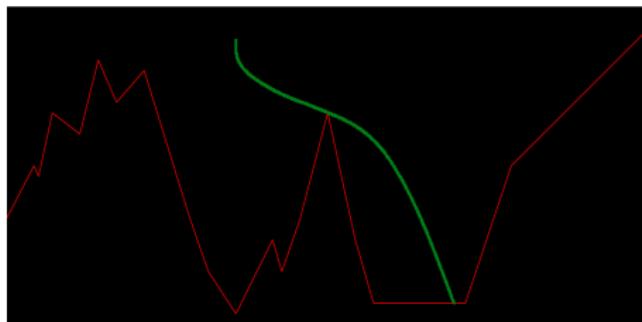


Figure – Sans zone de danger

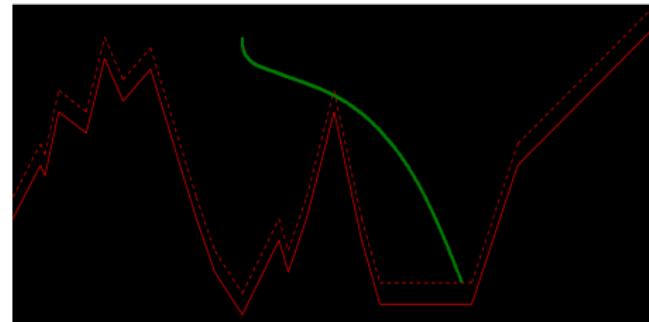


Figure – Avec zone de danger

## Zone de danger

Zone interdite permettant :

- d'éviter les cas limites dangereux pour la capsule.
- de négliger l'incertitude sur l'arrêt de la trajectoire (utilisation de flottants).

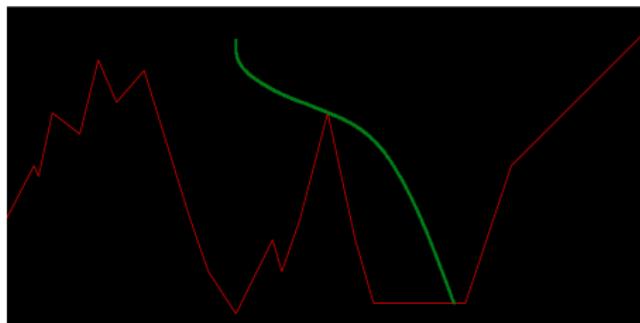


Figure – Sans zone de danger

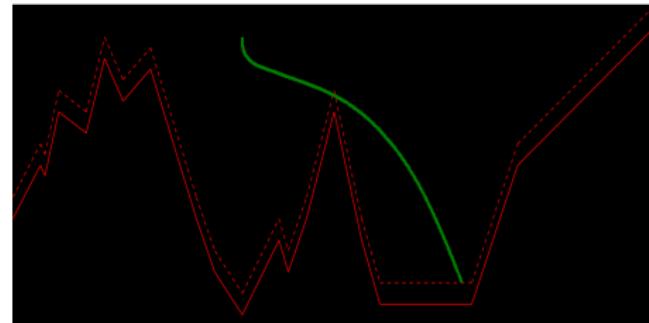
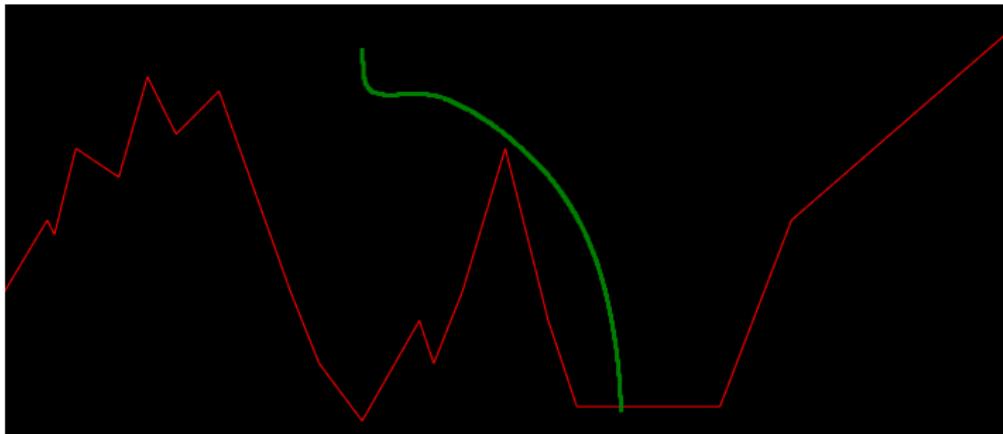
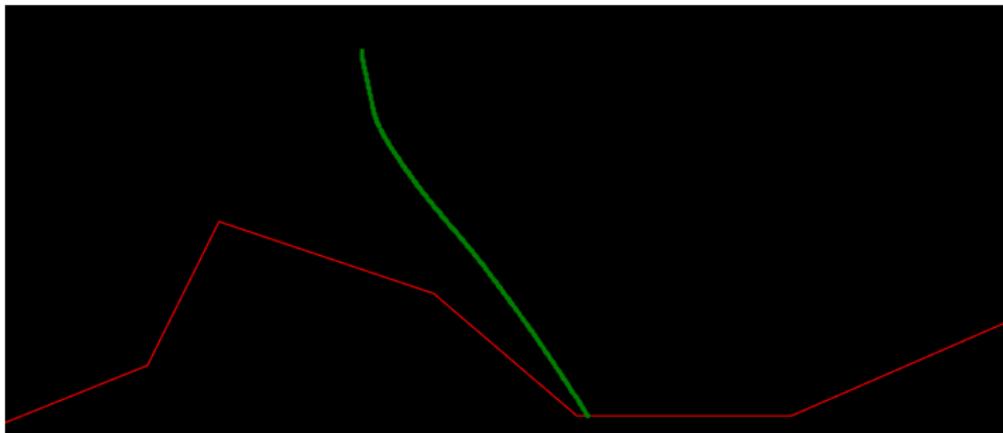
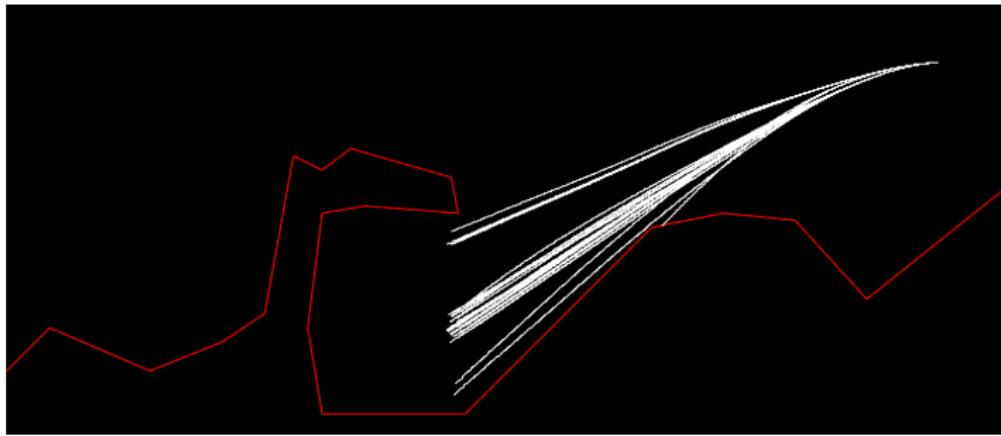
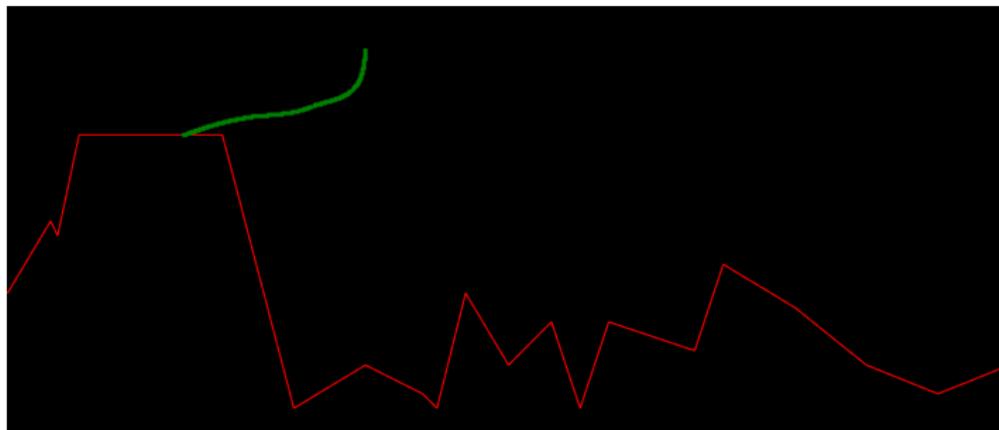
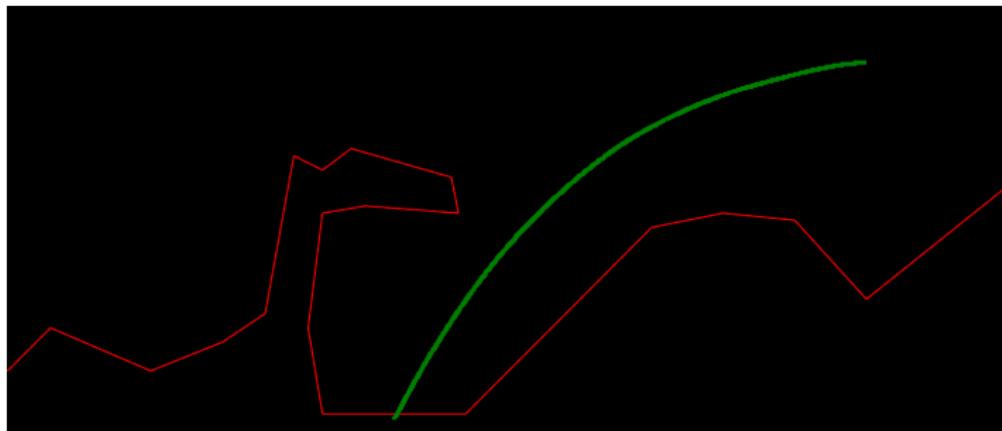


Figure – Avec zone de danger







# Sommaire

1 Introduction

2 Premières approches

3 Algorithme génétique

4 Conclusion

5 Annexe

# Sommaire

4

## Conclusion

- Apport de l'algorithme génétique
- Ouverture et améliorations

# Conclusion

## Résultats

- ① En théorie, si on ne limite pas le nombre de génération la propriété d'ergodicité assure la convergence (sous réserve d'existence d'une solution).
- ② L'algorithme dépend fortement du choix des opérateurs.
- ③ On a réussi à trouver des solutions avec une complexité temporelle en  $O(nmN)$ .

# Conclusion

## Résultats

- ① En théorie, si on ne limite pas le nombre de génération la propriété d'ergodicité assure la convergence (sous réserve d'existence d'une solution).
- ② L'algorithme dépend fortement du choix des opérateurs.
- ③ On a réussi à trouver des solutions avec une complexité temporelle en  $O(nmN)$ .

# Conclusion

## Résultats

- ① En théorie, si on ne limite pas le nombre de génération la propriété d'ergodicité assure la convergence (sous réserve d'existence d'une solution).
- ② L'algorithme dépend fortement du choix des opérateurs.
- ③ On a réussi à trouver des solutions avec une complexité temporelle en  $O(nmN)$ .

# Conclusion

## Résultats

- ➊ En théorie, si on ne limite pas le nombre de génération la propriété d'ergodicité assure la convergence (sous réserve d'existence d'une solution).
- ➋ L'algorithme dépend fortement du choix des opérateurs.
- ➌ On a réussi à trouver des solutions avec une complexité temporelle en  $O(nmN)$ .

Merci pour votre attention



# Sommaire

4

## Conclusion

- Apport de l'algorithme génétique
- Ouverture et améliorations

## Ouverture

- ① L'optimisation de la dépense en carburant peut être améliorée en utilisant un algorithme d'optimisation locale appliqué aux solutions obtenus.
- ② L'optimisation multiobjectif permettrait d'affiner la fitness tout en conservant les mêmes opérateurs mais nécessite de définir un voisinage propre à notre problème.

## Ouverture

- ① L'optimisation de la dépense en carburant peut être améliorée en utilisant un algorithme d'optimisation locale appliqué aux solutions obtenus.
- ② L'optimisation multiobjectif permettrait d'affiner la fitness tout en conservant les mêmes opérateurs mais nécessite de définir un voisinage propre à notre problème.

## Ouverture

- ① L'optimisation de la dépense en carburant peut être améliorée en utilisant un algorithme d'optimisation locale appliqué aux solutions obtenus.
- ② L'optimisation multiobjectif permettrait d'affiner la fitness tout en conservant les mêmes opérateurs mais nécessite de définir un voisinage propre à notre problème.

# Sommaire

1 Introduction

2 Premières approches

3 Algorithme génétique

4 Conclusion

5 Annexe

Dans cette annexe se trouve tout le code écrit en python. On trouvera dans l'ordre les modules suivants :

- Cinematique.py (fonctions permettant la représentation physique du programme)
- Genetique.py (fonctions utiles à l'algorithme génétique)
- Donnee.py (données du problème et paramètre)
- Main.py (programme principal : fonction d'évolution et interface graphique)

Dans cette annexe se trouve tout le code écrit en python. On trouvera dans l'ordre les modules suivants :

- Cinematique.py (fonctions permettant la représentation physique du programme)
- Genetique.py (fonctions utiles à l'algorithme génétique)
- Donnee.py (données du problème et paramètre)
- Main.py (programme principal : fonction d'évolution et interface graphique)

Dans cette annexe se trouve tout le code écrit en python. On trouvera dans l'ordre les modules suivants :

- Cinematique.py (fonctions permettant la représentation physique du programme)
- Genetique.py (fonctions utiles à l'algorithme génétique)
- Donnee.py (données du problème et paramètre)
- Main.py (programme principal : fonction d'évolution et interface graphique)

Dans cette annexe se trouve tout le code écrit en python. On trouvera dans l'ordre les modules suivants :

- Cinematique.py (fonctions permettant la représentation physique du programme)
- Genetique.py (fonctions utiles à l'algorithme génétique)
- Donnee.py (données du problème et paramètre)
- Main.py (programme principal : fonction d'évolution et interface graphique)

# Implémentation

```

def Trajectoire(m0, ind, dt, fuel, Lx, Lz): # O(n+C(intersection))
    """ Renvoie un tableau constitué des X et Z issu d'un individu (liste de couples (p,r) cf génétique) ainsi
que ses paramètres d'atterrissement avec m = [X, Z, Xspeed, Zspeed] """
    Lx, Lz = sol_securise(Lx, Lz)
    Lz = [-k for k in Lz]
    m = [k for k in m0] # copie profonde
    X, Z = m[0], m[1]
    Xspeed, Zspeed = m[2], m[3]
    tabX, tabZ = [X], [Z]
    Xp = X
    i = 2

    acc = pts_par_arc # changement de (p, r) lorsque acc = pts_par_arc: cela permet de respecter la contrainte
    # changement toutes les secondes et d'avoir plus de point pour la trajectoire

    while (0 <= X <= w) and (0 <= abs(Z) <= h) and (i < len(ind)) and fuel >= 0 and not Intersection(Xp, X, Z,
        Lx, Lz):

        if acc == pts_par_arc: # lisser la trajectoire ==> meilleure approximation de l'arrêt
            power, rotate = ind[i][0], ind[i][1]
            i += 1
            fuel -= power # modélise la consommation en fuel
            acc = 0
        acc += 1
        Xspeed, Zspeed = Vitesse(power, rotate, Xspeed, Zspeed, dt/pts_par_arc)
        X, Z = Eq_Horaires(power, rotate, Xspeed, Zspeed, X, Z, dt/pts_par_arc)
        Xp = tabX[-1]
        tabX.append(X), tabZ.append(Z)
        m = [X, Z, Xspeed, Zspeed]

    return tabX, tabZ, m, fuel, i # i est l'indice d'atterrissement

```

```

def Intersection(Xp, X, Z, Lx, Lz):
    """ Materialise le sol et donne des conditions d'arrêt quand on a un point en dessous du sol """
    ptc = []
    Lz = [-k for k in Lz]
    def segment(X, i):
        a = ((Lz[i]-Lz[i+1])/(Lx[i]-Lx[i+1])) # on a multiplié par -1 pour l'axe z
        b = (Lz[i]*Lx[i+1]-Lz[i+1]*Lx[i])/(Lx[i+1]-Lx[i])
        return (a*X+b, a, b)

    for i in range(len(Lx)-1):
        if (Lx[i] <= X < Lx[i+1]):
            ptc.append(i)

    if len(ptc) == 1:
        return abs(Z) <= segment(X, ptc[0])[0]
    elif len(ptc) == 2:
        i,j = ptc
        for k in ptc:
            if min(Lz[k], Lz[k+1]) <= abs(Z) <= max(Lz[k], Lz[k+1]) and abs(segment(X, k)[1]) > 1:
                z, a, b = segment(X, k)
                return (Xp <= (z-b)/a <= X) or (X <= (z-b)/a <= Xp) and abs(Z) <= z
        if max(Lz[i], Lz[i+1]) <= min(Lz[j], Lz[j+1]):
            return not (segment(X, i)[0] <= abs(Z) <= segment(X, j)[0])
        elif max(Lz[j], Lz[j+1]) <= min(Lz[i], Lz[i+1]):
            return not (segment(X, j)[0] <= abs(Z) <= segment(X, i)[0])

    return (not (segment(X, i)[0] <= abs(Z) <= segment(X, j)[0])) or (not (segment(X, j)[0] <= abs(Z) <= segment(X, i)[0]))

    else:
        print("zone non accessible")

```

```

# Initialisation #-----#
def rdmpower(power): # O(1)
    """ Renvoie une nouvelle poussée différente de au plus 1 m.s-2 avec l'ancienne """
    if power > p_min and power < p_max:
        return power + rdm.randint(-p_var, p_var)
    elif power == p_max:
        return power + rdm.randint(-p_var, 0)
    else:
        return power + rdm.randint(0, p_var)

#-----#
def rdmrotate(rotate): # O(1)
    """ Renvoie un angle différent de au plus π/12 avec l'ancien """
    if rotate >= radians(-(r_max-r_var)) and rotate <= radians(r_max-r_var):
        return rotate + radians(rdm.randint(-r_var, r_var))
    elif rotate > radians(r_max-r_var) and rotate <= radians(r_max):
        return rotate + radians(rdm.randint(-r_var, 0))
    else:
        return rotate + radians(rdm.randint(0, r_var))

#-----#
def creer_individu(p0, r0): # O(n)
    """ Génère un individu qui respecte les contraintes à partir d'une condition initiale (à t=0) """
    individu = [0]+[[p0, r0]]*nb_gene # le premier élément de la liste est l'indice d'atterrissement initialisé à 0
    for k in range(1, nb_gene-1):
        individu[k+1] = [rdmpower(individu[k][0]), rdmrotate(individu[k][1])]
    return individu

#-----#
def creer_population(p0, r0): # O(n*m)

    pop = [None]*taille_population
    for k in range(taille_population):
        pop[k] = creer_individu(p0, r0)
    return pop

```

```
# Mutations #-----
```

```
def mutation_individu(ind): # O(n)
    """ Modifie aleatoirement un gene en respectant les contraintes du mars lander """
    if ind == []:
        return []
    n = len(ind)
    for i in range(2, n-1): # indice des éléments modifiés: on ne modifie pas la CI
        if rdm.random() < proba_de_mutier:
            p, r = ind[i][0], degrees(ind[i][1])
            pl, p2 = ind[i-1][0], ind[i+1][0]
            r1, r2 = degrees(ind[i-1][1]), degrees(ind[i+1][1])
            pl, rl = [p], [r] # on garde intentionnellement la poussée non muté car il y a des cas où la mutation est impossible (ex: [2,3,4])
            for k in range(-p_max, p_max+1):
                if (p1-1 <= p+k <= p1+1) and (p2-1 <= p+k <= p2+1) and p+k != r:
                    pl.append(p+k)
            for k in range(-r_max, r_max+1):
                if (r1-1 <= r+k <= r1+1) and (r2-1 <= r+k <= r2+1):
                    rl.append(r+k)
            ind[i] = [rdm.choice(pl), radians(rdm.choice(rl))]
```

---

```
#-----
```

```
def mutation_population(pop): # O((n+C(traj))*m) --> # O(n*m)
```

```
for k in range(taille_population):
    mutation_individu(pop[k])
    if pop[k][0] != 0:
        i = pop[k][0][4]
        if abs(degrees(pop[k][i][1])) <= r_var: # on fait muter dès que possible l'angle final du lander afin d'obtenir une solution plus rapidement
            pop[k][i][1] = 0
```

```

# Croisement #-----  

def crossover_discret(ind1, ind2): # O(n) (complexité spatiale plus importante que crossover: O(n^2))  

    """ Croise deux individus en assurant une continuité, on essaie d'abord de couper en deux à l'indice i,  

    puis on regarde si on peut avec i-1 ou i+1, etc (methode discrete)"""  

    n = len(ind1)  

    i = n//2  

    croisements_posibles = []  

    fils = []  

    def criteres_croisement(i, ind1, ind2):  

        return (abs(ind1[i][0]-ind2[i][0]) <= 1) and (abs(ind1[i][0]-ind2[i][0]) <= radians(15))  

    for k in range((n//2)-1):  

        if criteres_croisement(i-k, ind1, ind2):  

            croisements_posibles.append(i-k)
            fils = ind1[:i-k] + ind2[(i-k):]
        if criteres_croisement(i+k, ind1, ind2):  

            croisements_posibles.append(i+k)
            fils = ind1[:i+k] + ind2[(i+k):]  

    nb_croisements_posibles = len(croisements_posibles)
    if nb_croisements_posibles > 1:  

        fils = ind1
        nb_croisements = rdm.randint(1, nb_croisements_posibles)  

        for j in range(1, nb_croisements-1):
            fils = fils[:j-1] + ind2[j-1:j] + fils[j:]
    return fils  

#-----  

def crossover(ind1, ind2): # O(n)  

    """ Methode de croisement avec float on prend le barycentre de deux individus (methode continue) """  

    b = rdm.random()
    n = len(ind1)
    fils1 = [0]+[None]*(n-1)
    fils2 = [0]+[None]*(n-1)
    for i in range(1, n):
        fils1[i] = [round(b*ind1[i][0]+(1-b)*ind2[i][0]), b*ind1[i][1]+(1-b)*ind2[i][1]]
        fils2[i] = [round((1-b)*ind1[i][0]+b*ind2[i][0]), (1-b)*ind1[i][1]+b*ind2[i][1]]
    return fils1, fils2

```

```

def evaluate(ind): # C(traj)
    """ Attribue un score inversement proportionnel à la distance à la zone d'atterrissement et
    l'angle/vitesse d'arrivé que l'on cherche à maximiser. Renvoie un boléen selon si l'individu
    est éligible à être solution """
    X, Z, m, fuel_f, i = cm.Trajectoire(m0, ind, dt, fuel, Lx, Lz)
    ind[0] = [X, Z, m, fuel_f, i]
    Xc, Zc = X[-1], Z[-1] # point après collision/atterrissage
    Xspeed, Zspeed = m[2], m[3]
    mid_zone = (zone[0]+zone[1])/2
    L = zone[1]-zone[0]
    rf = ind[i][1]

    note_distance = 100*((abs(Xc-mid_zone))/(abs(X0-mid_zone)))
    # rapport distance au milieu de la zone d'atterrissement par rapport sur distance initiale

    #note_distance = distance_au_sol(Xc, Zc, Lx, Lz)
    note = 500
    if (zone[0]+L*0.05 <= Xc <= zone[1]-L*0.05):

        if abs(Xspeed) >= v_speed_max:
            note -= 100*(abs(Xspeed)/300) # on pose une vitesse maximale à partir de laquelle le modèle n'est
plus valide
        if abs(Zspeed) >= h_speed_max:
            note -= 100*(abs(Zspeed)/300)
        if abs(degrees(rf)) >= r_var:
            note -= 100*(abs(round(degrees(rf)))/90)
    note -= note_distance
    note /= 5

    est_sol = (zone[0]+L*0.05 <= Xc <= zone[1]-L*0.05) and cm.collision(Xspeed, Zspeed, rf) and Z[-1] <= H <=
Z[-2] # on resserre la zone d'atterrissement de maniere arbitraire pour éviter les cas limites, génants en pratique
    return note, est_sol

```

```
def evolution(pop):
    """ Passage de la population n à n+1 """

    npop = evalue_population(pop)
    somme_note = 0
    solution = []
    pop_evaluee = [None]*taille_population
    k = 0

    # On separe les notes et les individus, de plus on donne un criterie de convergence

    for note, ind, est_sol in npop:
        #note = sharing(note, ind, npop)
        somme_note += note
        pop_evaluee[k] = ind
        if est_sol: # Le lander est sur la zone d'atterrisage et respecte les contraintes
            solution.append(ind)
        k += 1

    note_moyenne = somme_note/taille_population
    note_max = npop[0][0]

    if solution:
        return pop_evaluee, [note_moyenne, note_max], solution

    """ Methode élitiste: """
    # On conserve les individus les mieux notés et un nombre aleatoire d'individus plus faibles pour eviter de converger vers un maximum local

    parents = pop_evaluee[:nb_grade_retenu]
    for ind in pop_evaluee[nb_grade_retenu:]:
        if rdm.random() < proba_non_grade_retenu:
            parents.append(ind)
```

```

"""
Roulette wheel """ # O(nlog(n))

roulette = [None]*taille_population

k = 0
for note, ind, est_sol in nnpop:
    roulette[k] = [(note/somme_note), k]
    k += 1
roulette = sorted(roulette, reverse = True) # on trie par ordre décroissant
for i in range(len(roulette)):
    for j in range(i+1, len(roulette)):
        roulette[i][0] += roulette[j][0]
while len(parents) < nb_parents:
    alea = rdm.random()
    for k in range(1, len(roulette)):
        if roulette[k][0] <= alea <= roulette[k-1][0]:
            parents.append(pop_evaluee[roulette[k][1]])

"""
Si la note moyenne ne varie pas (extrema local) on conserve les meilleurs individus et on génère une nouvelle population pour compléter les parents """

n = len(lnote_moyenne)
it = 0
if n >= 1 and sum(lnote_moyenne[n-6:])/lnote_moyenne[-1]*5) >= 0.95: # si les 5 dernières note moyenne ne
varie pas de plus de 5% (arbitraire)
    parents = parents[:nb_grade_retenue]
    new_pop = gt.creer_population(power0, rotate0)
    while len(parents) < nb_parents:
        parents.append(new_pop[it])
        it += 1

fils = []
while len(fils) < taille_population-len(parents):
    """
    croisement continu """

    ind1, ind2 = rdm.choice(parents), rdm.choice(parents)
    fils1, fils2 = gt.crossover(ind1, ind2)
    fils.append(fils1)
    fils.append(fils2)

fils = parents + fils

# On applique les mutations
gt.mutation_population(fils)

return fils, [note_moyenne, note_max], solution

```

```

def trace_evolution(item = None):
    """ Represente graphiquement les générations de populations """
    global freq
    global pop, id_anim, graph
    global lnote_moyenne, lnote_max
    if graph:
        for individu in graph:
            can.delete(individu)
    graph = []
    max_fuel = 0

    if int(cpt.get()) == 0:
        pop = [gt.creer_population(power0, rotate0), [], []]
    else:
        pop = evolution(pop[0])
        lnote_moyenne.append(pop[1][0])
        lnote_max.append(pop[1][1])

    if pop[2] or int(cpt.get()) >= nb_de_generation_max: # si il existe une solution ou si nombre de génération trop grand
        for sol in pop[2]:
            X, Z, m, fuel_f, i = sol[0]
            max_fuel = max(max_fuel, fuel_f)
            tab = zip(X, Z)
            g_sol = can.create_line(tab, smooth = "true", width = 3, fill = "green")
        if pop[2]:
            freq[int(cpt.get())] += 1
            #prop[taille_population] += 1
            print("Atterrissage réussi: ", fuel_f, "-->, max_fuel)
        else:
            freq[-1] += 1
            print("Echec de l'atterrissement")

    else:
        for ind in pop[0]:
            X, Z, m, fuel_f, i = cm.Trajectoire(m0, ind, dt, fuel, Lx, Lz)
            tab = zip(X, Z)
            graph.append(can.create_line(tab, smooth = "true", width = 1, fill = "white"))

    for individu in graph:
        individu

    cpt.set(int(cpt.get())+1)
    if not pop[2] and int(cpt.get()) < nb_de_generation_max:
        id_anim = app.after(10, trace_evolution, item)

```

```

app = Tk()
app.title("Project Mars Lander: T5")

can = Canvas(app, scrollregion = (0, -h, 0, -h), bg = "black", height = h, width = w) # on a déplacé l'origine avec scrollregion
can.pack(side = "top")

mon_menu = Menu(app)
app.config(menu = mon_menu)

# Affichage par défaut du niveau 1
Lx = [k * scale for k in Coord_X[0]]
Lz = [-k * scale for k in Coord_Z[0]]
for i in range(len(Lx)-1):
    can.create_line(Lx[i], Lz[i], Lx[i+1], Lz[i+1], width = 1, fill = "red")
Lxup, Lzup = sol_securise(Lx, Lz)
Lzup = [-k for k in Lzup]
for i in range(len(Lxup)-1):
    can.create_line(Lxup[i], Lzup[i], Lxup[i+1], Lzup[i+1], width = 1, fill = "red", dash = (4, 4))

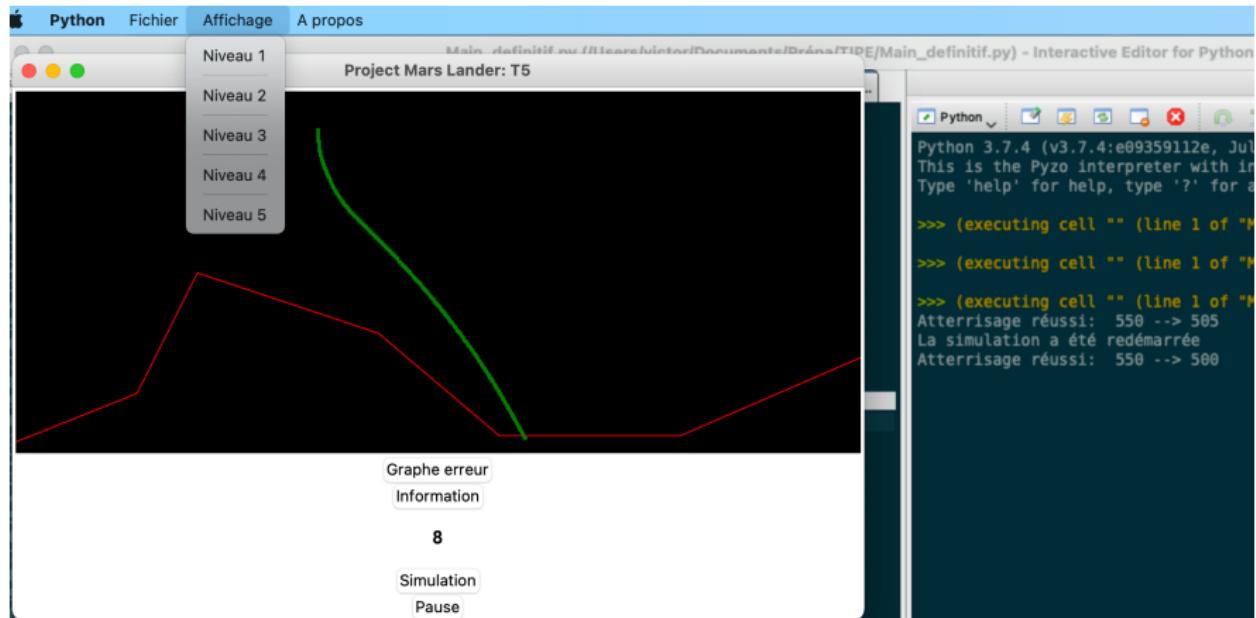
# Menu fichier
fichier = Menu(mon_menu, tearoff = 0)
fichier.add_command(label = "Redémarrer", command = restart)
fichier.add_separator()
fichier.add_command(label = "Quitter", command = quit)
mon_menu.add_cascade(label = "Fichier", menu = fichier)

# Menu cartes
cartes = Menu(mon_menu, tearoff = 0)
cartes.add_command(label = "Niveau 1", command = Niveau1)
cartes.add_separator()
cartes.add_command(label = "Niveau 2", command = Niveau2)
cartes.add_separator()
cartes.add_command(label = "Niveau 3", command = Niveau3)
cartes.add_separator()
cartes.add_command(label = "Niveau 4", command = Niveau4)
cartes.add_separator()
cartes.add_command(label = "Niveau 5", command = Niveau5)

mon_menu.add_cascade(label = "Affichage", menu = cartes)

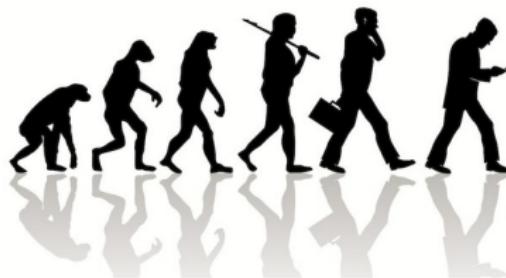
# A propos
apropos = Menu(mon_menu, tearoff = 0)
mon_menu.add_cascade(label = "A propos", menu = apropos)
apropos.add_command(label = "TTRP", command = a_propos)

```



## Résultats théoriques

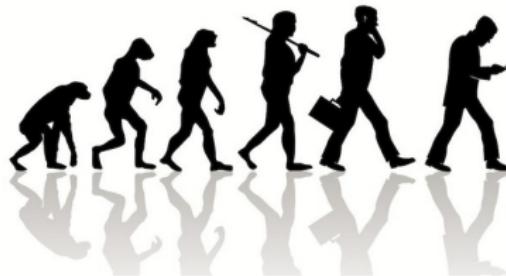
- ① Théorie des schémas de Goldberg.
- ② Convergence théorique (Modélisation basée sur les chaines de Markov de Cerf). (**admis**)
- ③ No Free Lunch Theorems : il n'existe pas de modélisation (opérateurs) qui s'adapte à tous les problèmes. (**admis**)



## Résultats théoriques

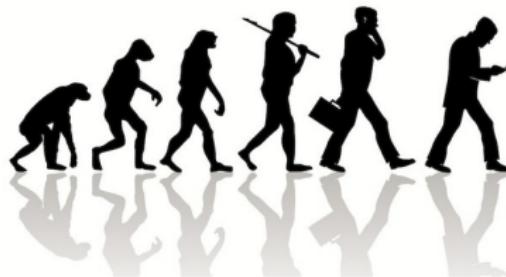
### ① Théorie des schémas de Goldberg.

- ② Convergence théorique (Modélisation basée sur les chaines de Markov de Cerf). (**admis**)
- ③ No Free Lunch Theorems : il n'existe pas de modélisation (opérateurs) qui s'adapte à tous les problèmes. (**admis**)



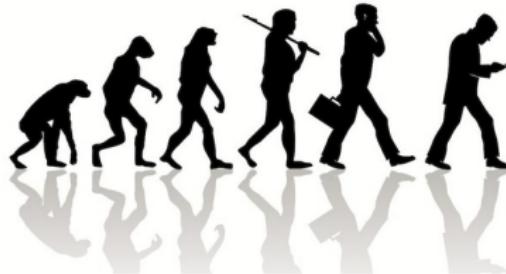
## Résultats théoriques

- ① Théorie des schémas de Goldberg.
- ② Convergence théorique (Modélisation basée sur les chaines de Markov de Cerf). (**admis**)
- ③ No Free Lunch Theorems : il n'existe pas de modélisation (opérateurs) qui s'adapte à tous les problèmes. (**admis**)



## Résultats théoriques

- ① Théorie des schémas de Goldberg.
- ② Convergence théorique (Modélisation basée sur les chaines de Markov de Cerf). (**admis**)
- ③ No Free Lunch Theorems : il n'existe pas de modélisation (opérateurs) qui s'adapte à tous les problèmes. (**admis**)



## Méthode 2 : Sélection par tournoi

On effectue un tirage avec remise de deux individus et on conserve celui avec la fitness la plus élevée avec une probabilité  $p \in [\frac{1}{2}, 1]$

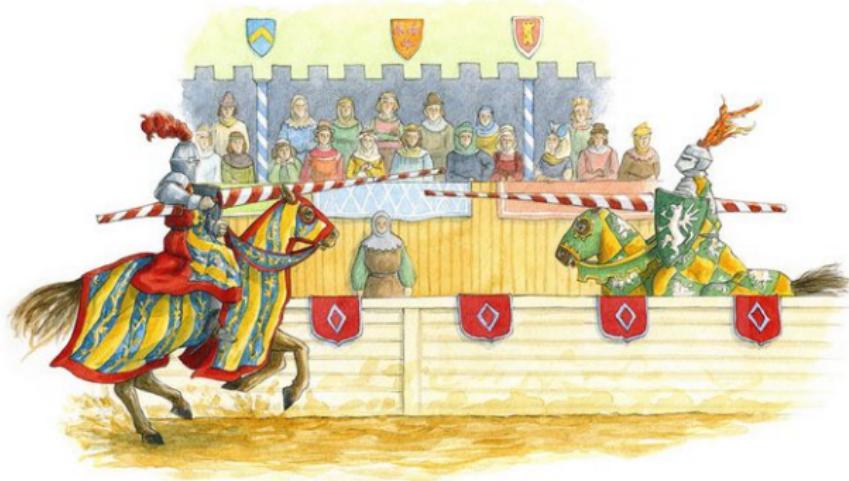


Figure – Tournoi (Google image)

# Comparaison de ces deux méthodes de sélection (50 échantillons)

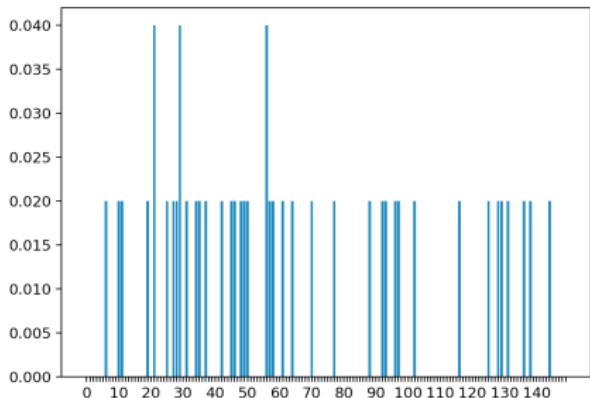


Figure – Roulette wheel

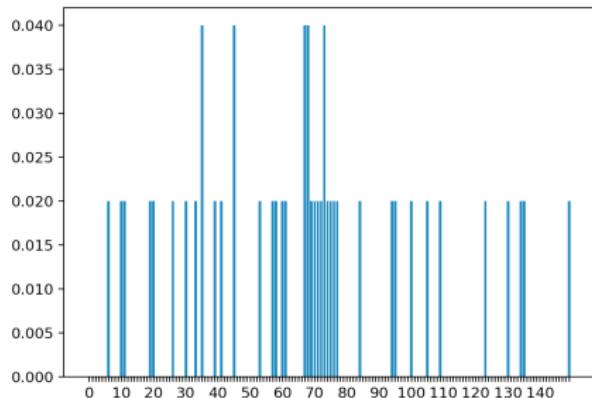


Figure – Sélection par tournoi

# Amélioration de la structure de donnée

## Structure

On stocke les informations sur la trajectoire de l'individu pour les calculer une fois seulement.

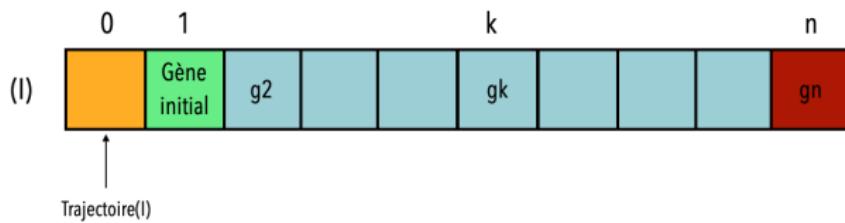
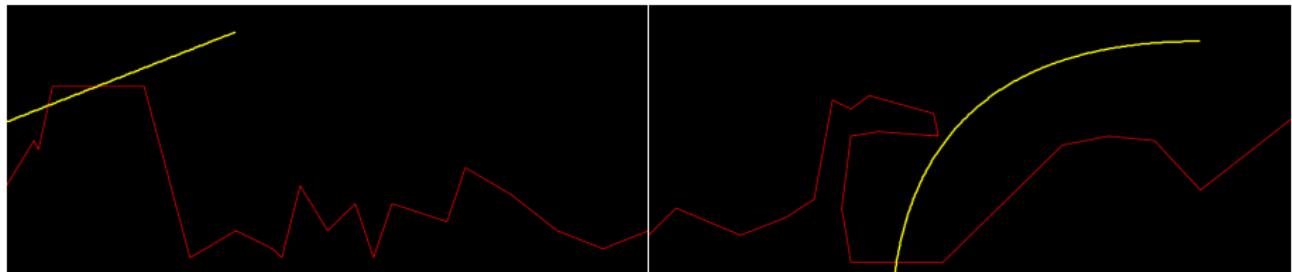
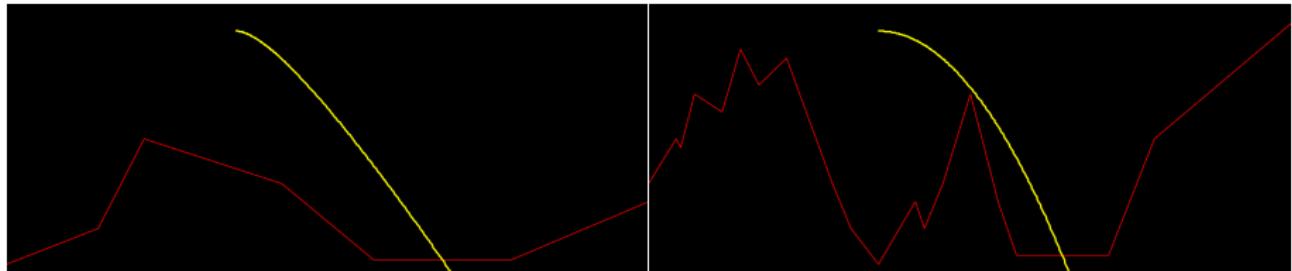


Figure – Représentation d'un individu

# Planification de trajectoire

Point de contrôle : position initiale - point le plus haut - zone d'atterrissage



# Optimisation de la fonction d'évaluation

## Scaling

L'objectif du scaling est d'amplifier artificiellement les écarts entre les individus.

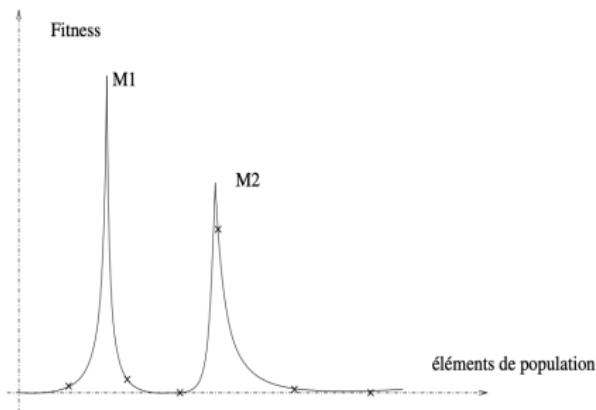


Figure – Cas adapté au scaling

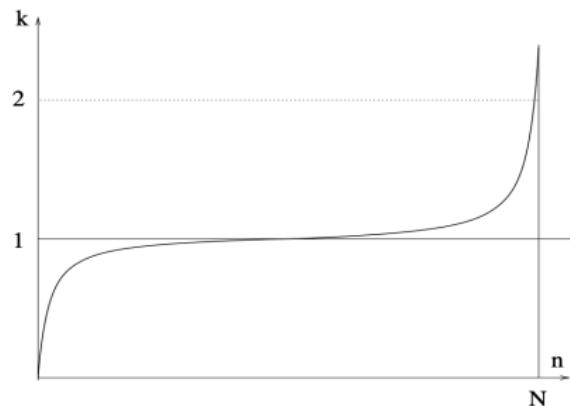


Figure –  $k = (\tan[(\frac{n}{N+1})\frac{\pi}{2}])^p$

## Sharing

L'objectif du sharing est de répartir les individus de la population sur tous les sommets de la fonction à optimiser.

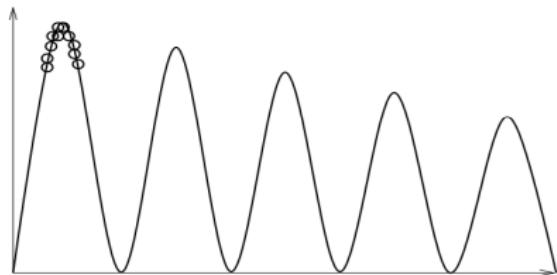


Figure – Sans sharing

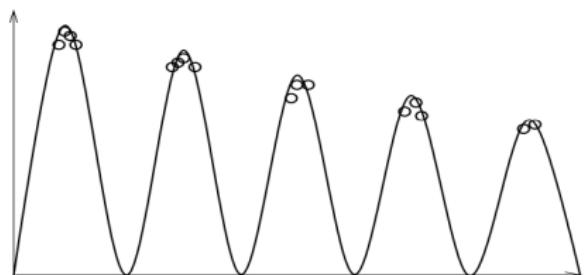


Figure – Avec sharing