Conduite autonome

Comment optimiser la batterie d'une voiture électrique ?

Maths en Jeans 2022–2023

ELOUADI Firdaous, SCOFIELD Alex, CHEVAILLIER Julien & RONSEAUX Aileen

Introduction

Pour accélérer, autrement dit, pour gagner de l'énergie cinétique, il est nécessaire de dépenser de l'énergie, qu'elle ait été produite par nos muscles, un moteur à combustion, ou qu'elle provienne d'une batterie électrique.

Lorsqu'une voiture thermique ralentit, son énergie cinétique est généralement perdue, tandis qu'une voiture électrique peut en récupérer une partie qu'elle stockera dans sa batterie.

Le *frein moteur* est le ralentissement naturel qui a lieu lorsque l'on cesse d'accélérer. Contrairement à ce que l'on appellera le *frein provoqué* – celui que l'on produit en appuyant sur la pédale de frein – il n'implique pas les disques de frein dont l'objectif est de créer des frottements pour causer une perte d'énergie cinétique. Le ralentissement est moins fort lorsqu'il est causé par le frein moteur mais il est celui qui permet de récupérer une quantité d'énergie maximale.

Notre problème est donc le suivant : quelle conduite adopter pour optimiser cette récupération d'énergie lors du ralentissement dans les situations suivantes ?

1. Rencontre d'un panneau stop : arrêt obligatoire au niveau du panneau.
2. Rencontre d'un feu tricolore : conduite à adapter en fonction de la couleur du feu.
3. Parcours d'une rue ponctuée d'une série de feux tricolores.

Cas 1 : panneau stop

Lorsque l'on arrive à un panneau stop, le code de la conduite nous impose un arrêt du véhicule au niveau de la balise. Ici, nous souhaitons effectuer cet arrêt en utilisant uniquement notre frein moteur, afin d'économiser au maximum l'énergie de notre véhicule.

Nous avons connaissance des éléments suivants :

a : l'accélération de notre véhicule, négative car nous ralentissons ;

t : la variable correspondant au temps écoulé depuis le début de l'observation ;

v0 : la vitesse initiale du véhicule ;

v(t) : la vitesse en fonction du temps, avec $v(t) = a \cdot t + v0$;

d(t) : la distance en fonction du temps, avec $d(t) = a \cdot t^2/2 + v0 \cdot t$ qui correspond à la distance parcourue depuis le début de l'observation.

Commençons par trouver combien de temps met la voiture à s'arrêter. Pour cela, on résout $v(t) = 0$ ce qui nous donne $t = -v0/a$.
On utilise ensuite cette valeur de t pour trouver la distance à parcourir pour s'arrêter grâce à la fonction d(t).
On trouve $d(-v0/a) = a/2 \cdot (-v0/a)^2 + v0 \cdot (-v0/a) = v0^2/2a - v0^2/a = -v0^2/2a$.

Ainsi, afin de pouvoir s'arrêter à temps et économiser un maximum d'énergie, il faut lever le pied de la pédale à $-v0^2/2a$ unités de distance de la balise stop.

Cas 2 : série de feux tricolores

Modélisation mathématique

Rue

Le premier élément essentiel de notre modélisation est la rue, sur laquelle seront disposés des feux ainsi qu'une voiture. Pour simplifier notre problème, la rue est rectiligne. Nous allons la représenter grâce à deux axes positifs : le temps en abscisse et la distance en ordonnée. La distance 0 représente le « point de départ » de la rue, et l'instant 0, l'instant du début de l'analyse du comportement de la voiture que nous étudions. Chaque élément sera représenté en fonction de sa position à un moment donné, selon sa nature et son état.

Selon l'avancement de notre étude, la rue sera soit de longueur finie, soit de longueur infinie.

Voiture

La voiture dont nous souhaitons étudier la consommation est décrite par sa position dans la rue en fonction du temps. La voiture ne peut pas reculer, mais peut s'arrêter. Ainsi, la voiture pourra être représentée par une fonction croissante, dont la dérivée première indiquera la vitesse, et la dérivée seconde l'accélération. Cette fonction sera appelée la *fonction de trajectoire* de la voiture, définie et continue sur un intervalle inclus dans $\mathbb{R}+$.

De plus, la voiture peut soit commencer son parcours à l'instant 0 soit plus tard. On utilisera le terme *instant de départ* pour désigner la borne inférieure de l'intervalle de définition de la fonction de trajectoire.

Nous considérons dans un premier temps que la variation de vitesse de la voiture est

instantanée. Sa fonction de trajectoire forme donc des angles et n'est pas dérivable en un nombre fini de points. Cela permet d'exprimer la fonction de trajectoire comme une fonction affine par intervalles, dont le coefficient directeur indique la vitesse de la voiture sur l'intervalle donné.

Notons que cela ne permet plus de représenter le frein moteur. On pourra néanmoins remarquer qu'il suffit d'arrondir la courbe pour en obtenir une représentation de plus en plus fidèle, similaire à celles que l'on a pu obtenir dans les deux premières parties du problème. On pourra alors considérer la tangente aux points d'inflexions pour calculer le gain d'une telle fonction, comme nous le verrons ultérieurement.

Feux tricolores

La situation que nous allons analyser consiste en une série de feux tricolores. C'est-à-dire un enchaînement de feux qui, au fil du temps, seront soit au rouge, soit au vert. Afin d'exprimer leur état (vert ou rouge) au fil du temps, nous avons besoin de décrire un feu par les propriétés suivantes.

- La *distance du feu* dans la rue, notée d, autrement dit la distance qui sépare le feu du point de distance 0.
- La *période du feu*, notée P. On suppose dans notre expérience que les feux ont une durée au vert et au rouge identiques. Une période représente donc la durée d'un cycle rouge-vert. On déduit de cette période la durée du feu au vert, et la durée du feu au rouge, égale à la moitié de la période du feu.
- L'*instant initial du feu*, c'est-à-dire l'instant auquel il commence pour la première fois un cycle rouge-vert. On le désignera par la lettre i avec $0 \leq i < P$. Par exemple, s'il est au rouge pour la première fois 3 secondes après l'instant zéro, alors $i = 3$. S'il est au rouge depuis 4 secondes à l'instant 0, on aura $i \equiv -4\ [P]$ soit $i = P - 4$ si $P \geq 4$.

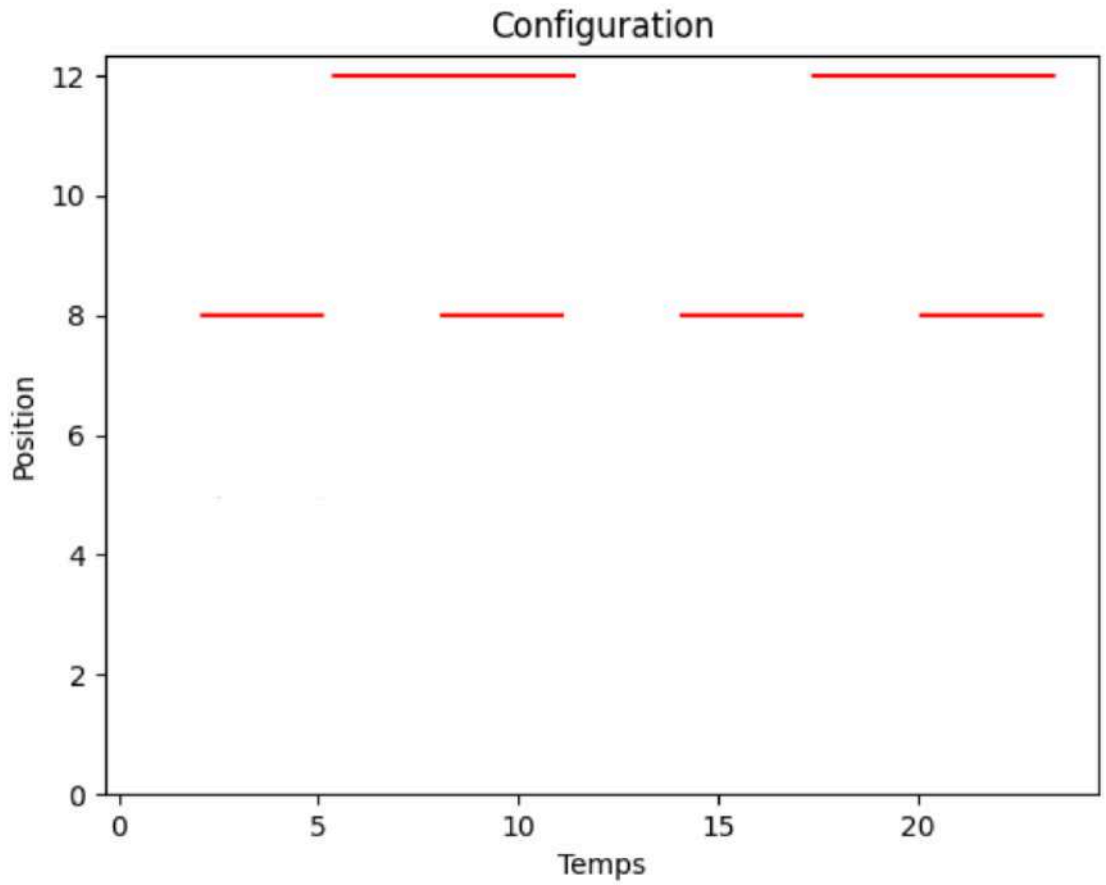
Nous désignerons par *configuration de feux* l'ensemble des feux d'une route, chacun caractérisé par ses attributs respectifs.

Soit un feu de période P, de distance d et d'instant initial i. Le feu sera représenté graphiquement comme une fonction de valeur constante d définie sur l'union d'intervalles $\{[k \cdot P + i\ ;\ (k \cdot P + i) + P/2[\mid k \in \mathbb{N}\} \cap \mathbb{R}+$. Ainsi, l'on représente les périodes où le feu est au rouge. Il ne doit y avoir aucun point d'intersection entre la fonction des feux et celle de la trajectoire de la voiture pour qu'aucun feu rouge ne soit grillé.

Exemples de représentations graphiques

Rue et feux

Voici l'exemple d'une rue sur laquelle se trouvent deux feux tricolores. Il nous est ainsi possible de savoir, à un instant donné, l'état (au vert ou au rouge) de chaque feu.



Le feu en haut du graphique est :

- de période $P = 10$ car il faut 10 secondes au feu pour effectuer un cycle rouge-vert ;
- de distance $d = 12$ car il vaut 12 sur les intervalles sur lesquels il est défini ;
- d'instant initial $i = 5$ secondes car la première phase rouge commence à 5 secondes.

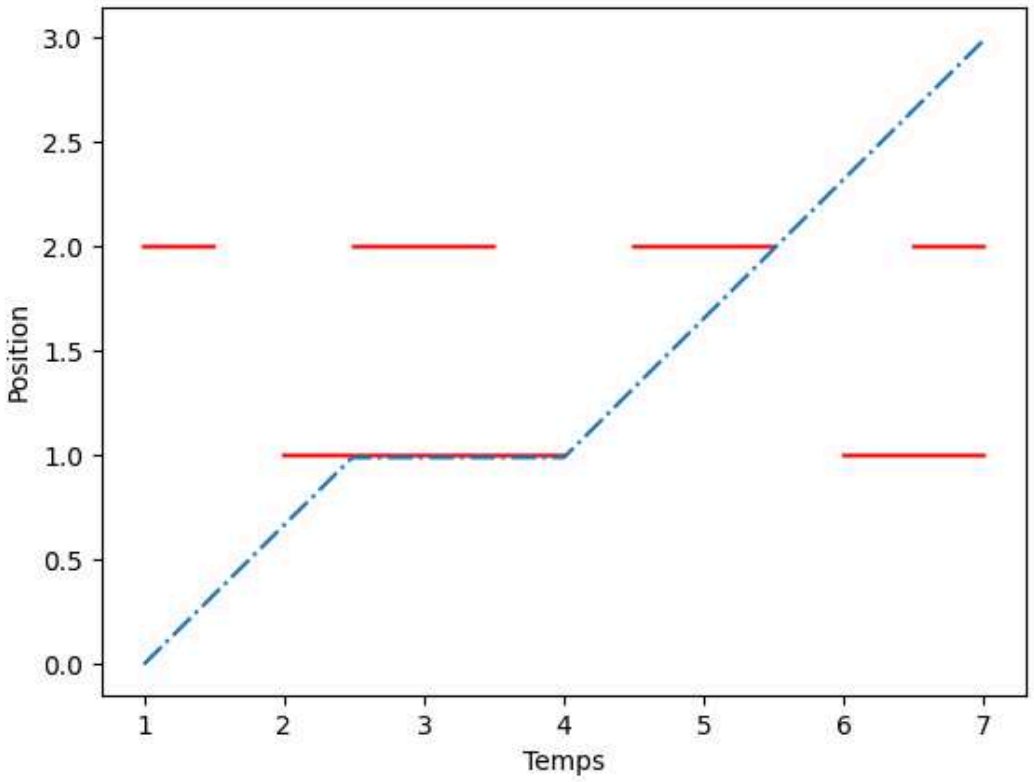
Le feu au milieu du graphique est :

- de période $P = 5$ car il faut 5 secondes au feu pour effectuer un cycle rouge-vert ;
- de distance $d = 8$ car il vaut 8 sur les intervalles sur lesquels il est défini ;
- d'instant initial $i = 2.5$ secondes car la première phase rouge commence à 2.5

secondes.

Rue, feux et voiture

Voici l'exemple d'une rue sur laquelle sont représentés deux feux et le trajet d'une voiture.



Dans cette situation, la voiture :

1. avance à vitesse constante ;
2. s'arrête au premier feu ;
3. reprend sa route à la même vitesse ;
4. passe le deuxième feu exactement à l'instant où il passe au vert ;
5. arrive au bout de la rue.

Comment optimiser l'énergie dépensée

Afin d'optimiser l'énergie dépensée par la voiture, nous allons chercher à quantifier l'énergie électrique gagnée ou perdue par la batterie lors d'une accélération ou une décélération (accélération négative).

On considérera qu'il n'y a pas de perte lors de la conversion de l'énergie électrique vers

l'énergie cinétique. Ainsi, si la batterie perd une quantité d'énergie E, i.e. gagne une quantité d'énergie électrique $-E$, alors la voiture gagnera E d'énergie cinétique.

Au contraire, l'énergie cinétique d'un ralentissement est pour la plupart perdue, et l'on considérera que seul $1/n$ de l'énergie sera conservée, $n \in \mathbb{N}^*$. Ainsi, la batterie gagnera une quantité d'énergie électrique $1/n$ si la voiture perd une quantité d'énergie 1.

L'on obtient ainsi un système de bonus-malus.

	Accélération	Décélération
Énergie électrique (*Ee*)	-1	$+ 1/n$
Énergie cinétique (*Ec*)	$+ 1$	-1
Facteur *F* = *Ee* / *Ec*	-1	$-1/n$

Ainsi, pour F le facteur correspondant à la situation (accélération ou décélération), si une voiture gagne une quantité d'énergie cinétique (positive ou négative) Ec, alors la batterie se rechargera de $Ee = F \cdot Ec$.

Pour quantifier cette énergie cinétique, nous utiliserons la formule suivante : l'énergie cinétique Ec d'un corps de masse m et de vitesse v est $Ec = \frac{1}{2}\, m \cdot v^2$. Considérons une voiture passant d'une vitesse initiale Vi à une vitesse finale Vf. L'énergie cinétique gagnée est donc la différence entre l'énergie cinétique finale et l'énergie cinétique initiale. Ainsi, la voiture aura gagné $\frac{1}{2}\, m\, (Vf^2 - Vi^2)$ d'énergie cinétique.

Ainsi, nous obtenons la fonction de gain g suivante.

$g(Vi, Vf) \quad = -\frac{1}{2}\, m\, (Vf^2 - Vi^2)$ en situation d'accélération, i.e. si $Vf \geq Vi$

$\qquad\qquad = -1/n \cdot \frac{1}{2}\, m\, (Vf^2 - Vi^2)$ si $Vf \leq Vi$

Note : nous utiliserons parfois le terme « coût » au lieu de « gain ». Ces deux désignations restent correctes car leurs valeurs sont simplement opposées (un gain de –3 représente un coût de +3).

Nous calculons ensuite, pour chaque variation de vitesse, le gain associé afin d'obtenir le *gain d'une fonction de trajectoire*. Optimiser la consommation d'une voiture électrique, c'est chercher à maximiser le gain ainsi obtenu.

Coût en énergie d'une configuration de feux

À présent, plaçons-nous en tant qu'architecte. Imaginons que nous souhaitons optimiser

la configuration des feux de manière à minimiser le coût moyen du passage des voitures dans la rue.

Différentes configurations de feux seront comparables à condition de considérer que toutes les voitures se comportent de manière identique. L'attitude que nous avons dans la vie quotidienne peut se décrire comme suit.

Algorithme 1

Répéter en boucle :

- s'il y a un feu et qu'il est rouge : s'arrêter ;
- sinon : rouler à la vitesse maximale réglementée.

Cela est dû au fait que nous ne savons pas quand le feu changera de couleur. Supposons donc que nous nous trouvons dans une ville où le temps avant le changement d'état du feu est affiché. Nous supposons que nous pouvons voir l'état du feu et son compteur à n'importe quel endroit dans la rue. Cela nous permet de créer l'algorithme de déplacement comme suit.

Algorithme 2

Répéter en boucle :

- s'il y a un feu en vue : adapter sa vitesse de sorte à passer le feu au vert (on optera pour la plus grande vitesse permettant un tel résultat sans dépasser la vitesse maximale réglementée) ;
- sinon : rouler à la vitesse maximale réglementée.

En choisissant un de ces algorithmes, nous permettons, pour une configuration de feux et un instant donnés, de créer une unique fonction de parcours.

De plus, nous pouvons calculer le gain d'une fonction de parcours basée sur les variations de cette fonction et la fonction de gain présentée précédemment.

Cela nous permet de créer une fonction de gain qui à chaque instant t associe le gain de la fonction de parcours d'instant de départ t.

Ainsi, le gain moyen d'une configuration de feux est la valeur moyenne de la fonction de gain que nous avons pu créer à l'aide de l'algorithme choisi.

Nous choisirons arbitrairement le deuxième algorithme, car il minimise le coût d'un déplacement. En effet, dans le cas du premier algorithme, la vitesse de la voiture variera

entre 0 et *Vmax*, la vitesse maximale réglementée ; alors que dans le deuxième algorithme, la vitesse variera entre deux vitesses initiales et finales (respectivement *Vi* et *Vf)* comprises dans l'intervalle [0, *Vmax*]. Ainsi, la différence ($Vf^2 - Vi^2$) sera moins grande si nous agissons selon le deuxième algorithme, ce qui minimise le coût selon l'expression de la fonction de coût présentée dans la partie précédente et que nous rappelons ci-dessous :

$g(Vi, Vf) = -\frac{1}{2} m (Vf^2 - Vi^2)$ en situation d'accélération, i.e. si $Vf \geq Vi$

$= -1/n \cdot \frac{1}{2} m (Vf^2 - Vi^2)$ si $Vf \leq Vi$.

Coût moyen pour une rue avec un unique feu

Soit une rue ne comportant qu'un feu de distance *d*, de période *P*, et d'instant initial *i*.

Puisque la fonction du feu est périodique, il nous suffit, pour calculer le gain moyen de la configuration de feux, de considérer tous les instants de départ *t* compris dans un intervalle d'amplitude *P*. Afin de faciliter les calculs, nous considérerons *t* appartenant à l'intervalle suivant : $[j ; j + P[$ avec $j = i - d/Vr$.

Soit une voiture d'instant de départ *t*. D'après l'algorithme de déplacement que nous avons choisi, nous pouvons scinder le déplacement de la voiture en deux phases.

Phase 1

À l'instant *t*, la voiture se trouve par définition à distance 0. Sachant qu'il y a un feu en vue, la voiture doit adapter sa vitesse de sorte à atteindre la distance *d* à un instant où le feu est au vert. On notera *V1(t)* la vitesse de la voiture d'instant de départ *t* pendant la phase 1 qui durera jusqu'à ce que la voiture atteigne la distance *d*.

Phase 2

Une fois le feu passé, aucun feu n'est en vue, et la voiture roulera à la vitesse maximale de réglementation *Vr*.

Ainsi, le gain total de la fonction de trajectoire de la voiture d'instant de départ *t* vaut $g(V1(t), Vr)$. Le gain moyen de cette configuration de feux est : $\frac{\int_j^{j+P} g(V1(t), Vr)\, dt}{P}$.

Trouvons une expression pour *V1(t)*. Nous rappellons que la vitesse *V1(t)* doit permettre à la voiture de passer au vert, et être maximale, tout en restant inférieure à la vitesse de réglementée *Vr*.

• Pour $t \in [j + P/2 ; j + P[$, on a $V1(t) = Vr$.

Pour montrer cela, calculons tout d'abord la durée Δt nécessaire à la voiture pour parcourir la distance d qui la sépare du feu, sachant qu'elle roule à la vitesse Vr. Selon la formule $v = d/t$, on obtient $\Delta t = d/Vr$. Vérifions donc que la voiture ne traversera pas un feu rouge si $t \in [j + P/2 ; j + P[$ et $V1(t) = Vr$.

Puisque $j = i - d/Vr$, on a : $i - d/Vr + P/2 \leq t < i - d/Vr + P$.

Or,

$$i - d/Vr + P/2 \leq t < i - d/Vr + P \quad \Leftrightarrow \quad i - d/Vr + P/2 + \Delta t \leq t + \Delta t < i - d/Vr + P + \Delta t$$

$$\Leftrightarrow \quad i - d/Vr + P/2 + d/Vr \leq t + \Delta t < i - d/Vr + P + d/Vr$$

$$\Leftrightarrow \quad i + P/2 \leq t + \Delta t < i + P.$$

Donc $i + P/2 \leq t + \Delta t < i + P$. Ainsi, la voiture atteindra le feu entre l'instant $i + P/2$ inclus et l'instant $i + P$ exclu. Or, comme i est l'instant initial du feu, le feu est au vert de l'instant $i + P/2$ inclus à l'instant $i + P$ exclu. V1(t) = Vr permet donc de passer au vert. De plus, la valeur $V1(t)$ est maximale car Vr majore $V1(t)$.

• Pour $t \in [j ; j + P/2[$, $V1(t) = d / (i + P/2 - t)$.

Nous pouvons calculer une vitesse comme le quotient d'une distance par une durée. Si l'on considère la distance d, il nous suffit de trouver une durée $\Delta t'$ telle que $t + \Delta t' \notin \{[k \cdot P + i ; (k \cdot P + i) + P/2[\mid k \in \mathbb{N}\} \cap \mathbb{R}+$, *i.e.* que le feu soit vert à l'instant $t + \Delta t'$.

Or,

$$i + P/2 \leq t + \Delta t' < i + P \quad \Leftrightarrow \quad i + P/2 - t \leq \Delta t' < i + P - t.$$

Donc $\forall \Delta t' \in [i + P/2 - t ; i + P - t[$, $v = d/\Delta t'$ permet de passer au vert. Néanmoins, $V1(t)$ doit être maximale, ainsi, nous cherchons à minimiser $\Delta t'$. Donc $V1(t) = d / (i + P/2 - t)$.

On peut vérifier en outre que $V1(t) \leq Vr$. En effet, $i - d/Vr \leq t < i - d/Vr + P/2$, et on a :

$$i - d/Vr \leq t < i - d/Vr + P/2 \quad \Leftrightarrow \quad -i + d/Vr \geq -t > -i + d/Vr - P/2$$

$$\Leftrightarrow \quad d/Vr + P/2 \geq -t + i + P/2 > d/Vr$$

$$\Rightarrow \quad -t + i + P/2 > d/Vr$$

$$\Rightarrow \qquad 1 / (i + P/2 - t) < Vr/d$$

$$\Rightarrow \qquad d / (i + P/2 - t) < Vr$$

$$\Rightarrow \qquad V1(t) < Vr.$$

Donc $V1(t) < Vr$.

Ainsi,

$V1(t) \quad = d / (i + P/2 - t) \qquad$ si $t \in [j ; j + P/2[$

$\qquad\quad = Vr \qquad\qquad\qquad$ si $t \in [j + P/2 ; j + P[$

D'après la formule, $g(Vi, Vf) = - ½ m (Vf^2 - Vi^2)$ si $Vf \geq Vi$. Comme $V1(t) \leq Vr$, on a donc $g(V1(t), Vr) = - ½ m (Vr^2 - V1(t)^2)$.

D'où :

$g(V1(t), Vr) \quad = - ½ m (Vr^2 - (d / (i + P/2 - t))^2) \qquad$ si $t \in [j ; j + P/2[$

$\qquad\qquad\quad = - ½ m (Vr^2 - Vr^2) = 0 \qquad\qquad$ si $t \in [j + P/2 ; j + P[$.

De plus,

$$\frac{\int_j^{j+P} g(V1(t), Vr)\, dt}{P} = \frac{\int_j^{j+P/2} g(V1(t), Vr)\, dt + \int_{j+P/2}^{j+P} g(V1(t), Vr)\, dt}{P} = \frac{\int_j^{j+P/2} g(V1(t), Vr)\, dt + 0}{P} = \frac{\int_j^{j+P/2} g(V1(t), Vr)\, dt}{P}.$$

Calculons donc l'intégrale $I = \int_j^{j+P/2} g(V1(t), Vr)\, dt$.

$$I = \int_j^{j+P/2} - ½ m (Vr^2 - (d / (i + P/2 - t))^2)\, dt$$

$$I = \int_j^{j+P/2} - ½ m Vr^2\, dt + \int_j^{j+P/2} ½ m d^2 / (i + P/2 - t)^2\, dt$$

$$I = [- ½ m Vr^2 \cdot t]_j^{j+P/2} + ½ m d^2 \cdot \int_j^{j+P/2} 1/(i + P/2 - t)^2\, dt$$

$$I = [- ½ m Vr^2 \cdot t]_j^{j+P/2} - ½ m d^2 \cdot \int_j^{j+P/2} - 1/(i + P/2 - t)^2\, dt$$

On reconnaît la forme $u' \circ v \cdot v'$ dont la primitive est $u \circ v$ avec :

$u(x) = -1/x$ et $u'(x) = 1/x^2$

$v(x) = i + P/2 - x$ et $v'(x) = -1$.

D'où :

$$I = [-\tfrac{1}{2} m Vr^2 \cdot t]_j^{j+P/2} - \tfrac{1}{2} m d^2 \cdot [1/(-i - P/2 + t)]_j^{j+P/2}$$

$$I = -\tfrac{1}{2} m Vr^2 \cdot (j + P/2) - (-\tfrac{1}{2} m Vr^2 \cdot j) - \tfrac{1}{2} m d^2 \, (1/(-i - P/2 + j + P/2) - 1/(-i - P/2 + j))$$

$$I = -1/4 \, m P Vr^2 - \tfrac{1}{2} m d^2/(-i + j) + \tfrac{1}{2} m d^2/(-i - P/2 + j)$$

Puisque $j = i - d/Vr$,

$$I = -1/4 \, m P Vr^2 - \tfrac{1}{2} m d^2/(-i + i - d/Vr) + \tfrac{1}{2} m d^2/(-i - P/2 + i - d/Vr)$$

$$I = -1/4 \, m P Vr^2 - \tfrac{1}{2} m d^2/(-d/Vr) + \tfrac{1}{2} m d^2/(-P/2 - d/Vr)$$

$$I = -1/4 \, m P Vr^2 + \tfrac{1}{2} m d \, Vr + \tfrac{1}{2} m d^2/(-P \, Vr/2Vr - 2d/2Vr)$$

$$I = -1/4 \, m P Vr^2 + \tfrac{1}{2} m Vr \, d - m Vr \, d^2/(P \, Vr + 2d)$$

On peut donc en déduire le gain moyen de la configuration de feux choisie, valant I/P :

$$I/P = \frac{-1/4 \, m P Vr^2 + \tfrac{1}{2} m Vr \, d - m Vr \, d^2/(P \, Vr + 2d)}{P}$$

$$= -1/4 \, m Vr^2 + m Vr \, d/2P - m Vr \, d^2/(Vr \, P^2 + 2dP)$$

Configuration de feux optimale

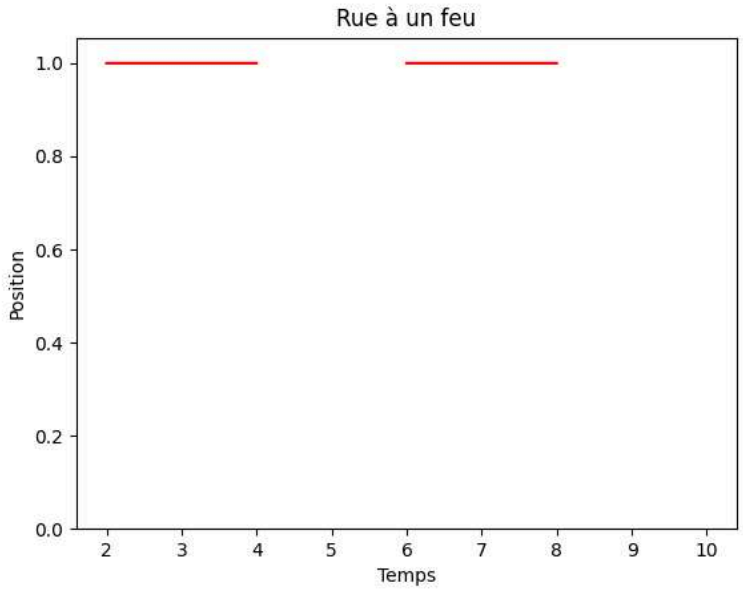
Une rue dont les feux sont synchronisés est une rue dans laquelle il est possible de ne croiser aucun feu rouge si l'on roule à la vitesse maximale réglementaire. En conséquence, si le premier feu que l'on croise dans une telle rue est vert et que notre vitesse est adaptée, alors on est certain que tous les feux de la rue seront verts au moment où on les franchira.

Ainsi, par définition, si on conduit dans une telle rue, on peut se retrouver face à 2 situations.
- Soit le premier feu que l'on croise est vert, alors on n'aura jamais besoin de faire varier notre vitesse, et le coût du parcours de la rue est ainsi nul.
- Soit le premier feu que l'on croise est rouge, alors, quel que soit l'algorithme choisi, le

coût de la rue se résumera au coût du premier feu, puisque dès le second feu on revient à la première situation.

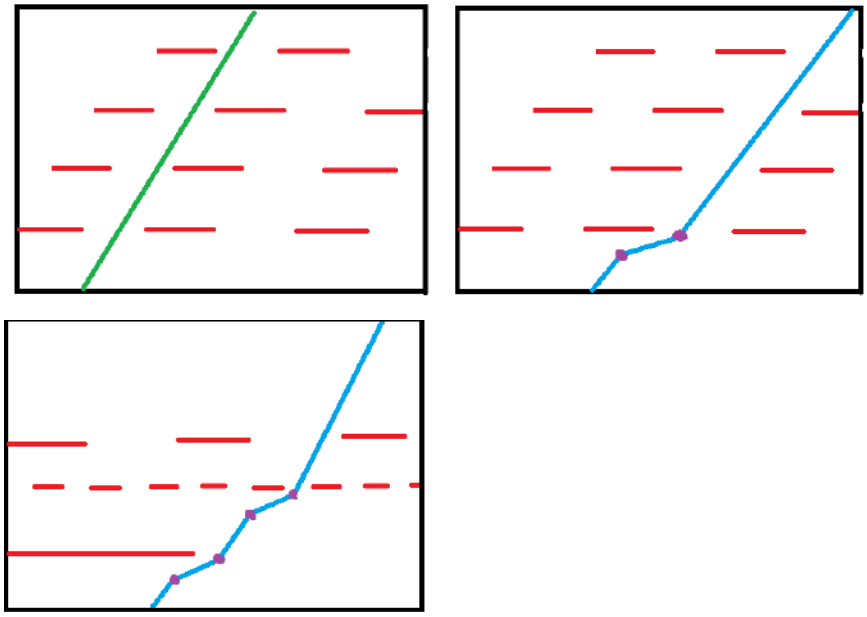
Dès lors, si les feux d'une rue sont synchronisés, alors, à partir du second feu, un conducteur pourra tous les franchir à vitesse constante, quel que soit l'instant auquel il arrive. De plus, comme un feu est vert aussi longtemps qu'il est rouge, le conducteur a une chance sur 2 que le premier feu soit rouge, donc, le coût moyen de parcours d'une telle rue sur un grand nombre d'itération sera de la moitié du coût de traverse d'un feu.

Nous allons désormais comprendre pourquoi cette configuration est nécessairement optimale.
Soit une rue dépourvue de feux tricolores. Nous en ajoutons un unique, et nous retrouvons donc avec une rue à un seul feu. Celui-ci n'a pas besoin de synchronisation : il est seul.

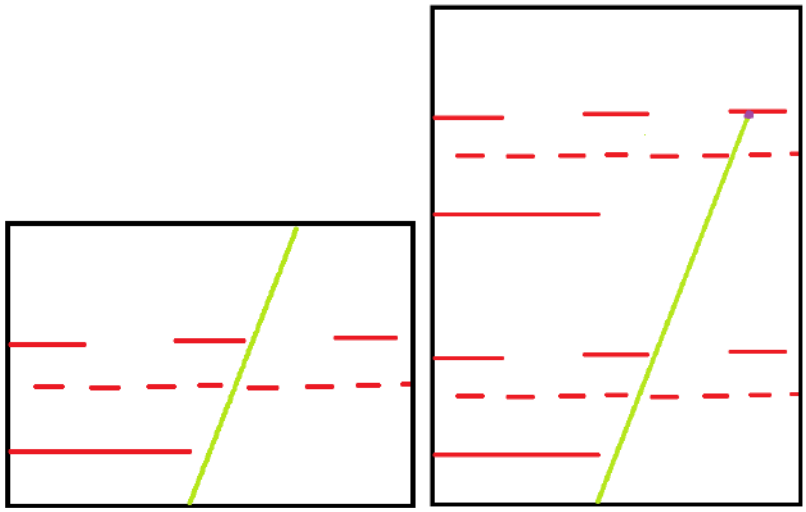


Ajoutons en un second, et constatons ce qu'il se passe selon qu'il soit synchronisé ou non avec le premier.

S'il est synchronisé avec le précédent, il n'y a rien de nouveau à observer : la rue est composée de feux synchronisés.



Cependant, s'il n'est pas synchronisé, on constate que le conducteur n'est pas garanti de pouvoir traverser le second feu quel que soit l'instant où il arrive, en maintenant sa vitesse constante, et le coût moyen de parcours augmente en conséquence.



Si nous continuons d'ajouter des feux, nous constaterons la même chose : Si un nouveau feu est synchronisé avec les précédents, tout se passera bien. Dans le cas contraire, la garantie qu'a le chauffeur de pouvoir franchir la rue en une seule fois sera compromise. La configuration en question aura donc un coût moyen plus élevé que la configuration synchronisée.

La configuration des feux synchronisés étant celle ayant le coût moyen le plus bas, elle

est donc la plus optimale.

Formalisation, périodicité et minimisation de coûts

Pour des situations plus complexes, ce n'est pas facile de trouver des minimums pour la fonction de coût (en fait, si on considère une voiture qui accélère de façon continue, il faut réviser une nouvelle fonction de coût). On a formalisé la situation de telle façon qu'on a trouvé des méthodes pour trouver des minimums locaux de stratégies, en faisant varier peu une stratégie. C'est-à-dire, on a trouvé une méthode pour perturber des stratégies pour réduire leur coût. Le détail de cette formalisation est donné en annexe, nous nous contentons ici d'en résumer les étapes.

Du point de vue de l'architecte, on a démontré trois résultats importants.

On a trouvé ce qu'on appelle les fenêtres de validité, qui donnent, pour une configuration donnée quelconque, les vitesses qui nous permettent de traverser, à vitesse constante, toute la rue, sans s'arrêter. On peut généraliser ce résultat à des rues infinies.

L'expression

$$\mathcal{F}_x[\{\varphi_i\}_{i\in I}] = \bigcap_{i\in I}\left(\bigcup_{k\in\mathbb{Z}}\left]\frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T}\right[\cap \mathbb{R}_+^*\right)$$

nous donne, pour des feux φ_i, les vitesses pour lesquelles on peut traverser la rue jusqu'au point x (ici φ_i^T dénote la période du i-ème feu, φ_i^θ sa phase φ_i^x sa position, supposée plus petite que x).

Le deuxième résultat intéressant est que l'espace des configurations périodiques est dense dans l'espace des configurations. C'est-à-dire, on peut approcher n'importe quelle configuration par des configurations périodiques.

Pour les configurations périodiques, on peut définir le coût moyen comme l'intégrale du coût des stratégies naturelles (c'est-à-dire suivre toujours la vitesse maximale, quand c'est possible), divisé entre la période. On a fourni un algorithme qui minimise cette magnitude.

Minimiser ce coût moyen est utile, car il est proportionnel à la consommation d'essence totale dans la rue, si on suppose qu'il y a des voitures en y entrant et en y sortant

uniformément. C'est-à-dire, plus le coût est petit, moins les voitures (en moyenne) doivent accélérer, et moins d'essence est dépensée.

On a aussi implémenté cette formalisation et divers algorithmes étudiés en Python.

Conclusion

La question initiale, qui se voulait très large, était sujette à de nombreuses interprétations et variations. Nous avons donc pu explorer de nombreuses pistes :

- en fixant tour à tour différents paramètres (accélération constante, durée des feux, etc.) ;
- en changeant les hypothèses avec lesquelles nous travaillons (hypothèse de synchronisation des feux, de l'infinité de la rue à parcourir, de l'espacement régulier des feux, etc.) ;
- en changeant les objectifs à atteindre (optimiser le passage d'une voiture, optimiser le passage des voitures en moyenne, etc.).

L'exploration de ces diverses pistes nous mène difficilement à un consensus ou à une règle générale. Cependant, nous pouvons souligner les observations suivantes.

L'hypothèse que le conducteur ou la conductrice du véhicule connaît le temps restant avant le changement de couleur du feu est essentielle dans le cadre de nos résultats. Cependant, cela pose deux problèmes concernant une application concrète de ces derniers :

- aucune indication temporelle n'accompagne généralement les feux tricolores (en France à minima) ;
- et les calculs à réaliser, même s'ils peuvent être simples, en particulier dans les premières applications (panneau stop et feu unique), sont trop complexes à réaliser de tête au cours de la conduite pour s'avérer pertinents.

Cependant, le concept de frein moteur reste intéressant, et si ces calculs sont nécessaires pour en imaginer une application optimale (passer au niveau du feu avec une vitesse maximale), on peut tout de même en obtenir un gain intéressant d'un point de vue énergétique que la voiture soit électrique ou thermique. En effet, comme le frein provoqué permet un ralentissement plus fort, la voiture aura roulé à vitesse maximale pendant plus longtemps, et aura donc gaspillé l'énergie utile pour maintenir cette vitesse maximale, ce que l'activation anticipée du frein moteur aurait évité.

Enfin, le passage des voitures est optimisé d'un point de vue individuel et collectif avec l'utilisation des feux synchronisés, et ne demande en outre pas de réflexion, calculs ou engagement individuel de la part des conducteurs et conductrices pour être efficace.

Ces deux derniers points pourront donc être des applications concrètes, à appliquer à un niveau individuel ou collectif, des résultats de ce sujet de recherche qui se voulait concret.

Annexe: Formalisation, périodicité et minimisation de coûts de configurations

April 7, 2024

1 Introduction

On est en voiture et on se trouve face à une rue. L'objectif est de trouver la meilleure stratégie pour traverser la rue de la façon la plus efficiente possible. On va proposer un modèle théorique pour décrire cette situation dans toute sa généralité, ainsi que des outils pratiques pour étudier une telle situation concrète.

Pour modéliser cette situation on va analyser deux éléments: la voiture et la configuration des feux. Il y a, donc, deux points de vue: celui de l'architecte et celui du conducteur. On va commencer par décrire le premier.

Définition 1.1 (Feu)**.** *On appelle* ***feu*** *(noté par φ) tout élément $\big(x, T, \theta, f\big)$, avec $x, T \in \mathbb{R}_+, \theta \in \mathbb{R}_+$ et $f : \mathbb{R}_+ \to \{0, 1\}$ qui vérifie:*

$$f(t) = \lfloor \frac{t}{T} + \theta \rfloor \ (mod\ 2)$$

On note avec $\varphi^x, \varphi^T, \varphi^\theta$ et φ^f les composantes de φ.

Cette définition caractérise de forme précise un feu. On remarque que x est sa position dans la rue et T son période. On a θ sa phase (sans elle tous les feux commenceraient dans le même état) et f une fonction qui nous dit si le feu est rouge ou vert en fonction du temps.

On rémarque que le feu passe la moitié du temps en vert et la moitié du temps en rouge. On pourrait modéliser des feux avec des périodes variables. Pour la plupart des résultats on ne s'interesse pas aux détails des fonctions φ^f. Dans les cas où ceci est important on le précisera explicitement.

Définition 1.2 (Configuration)**.** *Une* ***configuration*** *est une famille de feux (qu'on indexe avec un ensemble I) qui vérifie $\forall i, j \in I, (i \neq j) \implies (\varphi_i^x \neq \varphi_j^x)$.*

Définition 1.3 (État)**.** *Soit $\{\varphi_i\}_{i\in I}$ une configuration. On appelle* ***état*** *de $\{\varphi_i\}_{i\in I}$ au temps t, la famille: $\mathcal{E}(\{\varphi_i\}_{i\in I}, t) = \{\varphi_i^f(t)\}_{i\in I}$. Si la configuration est claire par le contexte, on écrira juste $\mathcal{E}(t)$.*

Dans les cas que l'on va étudier il va y avoir soit une quantité finie de feux ou une quantité dénombrable. C'est-à-dire, l'ensemble I ne va jamais être non-dénombrable.

Passons maintenant au point de vue du conducteur.

Définition 1.4 (Stratégie)**.** *Une stratégie est une fonction $V : \mathbb{R}_+ \to \mathbb{R}_+$ intégrable et dont l'intégrale est continue.*

On remarque que la stratégie n'est qu'un choix de fonction de vitesse pour la voiture, de telle façon qu'elle ne récule jamais. Cette permière définition de stratégie n'est pas très rigide, et, eventuellement, elle admet une acceleration discontinue et non bornée.

On peut définir la fonction $S : \mathbb{R}_+ \to \mathbb{R}_+$ avec $S(t) = \int_0^t V(\tau)d\tau$, par la définition de V. On remarque que l'ensemble d'arrivé est $\mathbb{R}_+$, par la positivité de l'intégrale.

Définition 1.5 (Stratégie valide)**.** *Soient $x_0, x_f \in \mathbb{R}_+$ avec $x_0 \leq x_f$. Une stratégie V est dite valide entre x_0 et x_f pour une configuration de feux $\{\varphi_i\}_{i\in I}$ si:*

1. *$\exists t \in \mathbb{R}_+$ vérifiant $S(t) = x_f$.*
2. *$\forall t \in \mathbb{R}_+$ avec $S(t) \in [x_0, x_f], \forall \varphi_i \in \{\varphi_i\}_{i\in I}$,*

$$\Big(S(t) = \varphi_i^x\Big) \implies \Big(V(t) = 0\Big) \vee \Big(\varphi_i^f(t) = 1\Big)$$

Dans la plupart des applications on s'intéressera au cas où $x_0 = 0$. Plus tard, quand on parlera de rues infinies, on aura besoin d'une autre version de stratégie valide, qu'on va définir ici aussi.

Définition 1.6 (Startégie infinie valide)**.** *Soit $x_0 \in \mathbb{R}_+$. Une stratégie est dite valide entre x_0 et $+\infty$ pour une configuration de feux $\{\varphi_i\}_{i\in I}$ si:*

1. *$\forall x \geq x_0, \exists t \in \mathbb{R}_+$ tel que $S(t) > x$.*
2. *$\forall t \in \mathbb{R}_+$ avec $S(t) \in [x_0, +\infty[, \forall \varphi_i \in \{\varphi_i\}_{i\in I}$,*

$$\Big(S(t) = \varphi_i^x\Big) \implies \Big(V(t) = 0\Big) \vee \Big(\varphi_i^f(t) = 1\Big)$$

On remarque que cette dernière condition se résume en la deuxième condition de la définition pour le cas fini, mais avec $x_f = +\infty$. La première condition nous dit que S n'est pas bornée. C'est-à-dire, il n'y a pas de points de la rue que la voiture ne vas pas surpasser.

Proposition 1.1. *Soient $x_0, x_f \in \mathbb{R}_+$ avec $x_0 \leq x_f$ et V une stratégie valide entre x_0 et x_f pour une configuration de feux données. La fonction S est surjective dans $[x_0, x_f]$.*

Démonstration. Par validité de V, on sait que $S^{-1}(x_f) \neq \emptyset$. On prends $t_f = \inf\big(S^{-1}(x_f)\big)$. On remarque que la fonction S est croissante sur $[0, t_f]$, puisqu'elle est une primitive de V, qui appartient à $\mathbb{R}_+$ sur cet intervalle. En plus, par continuité de V, S est continue. On peut conclure que S est injective sur $[0, t_f]$ à images sur $[0, x_f[$, et en prenant un élément de $S^1{-}1(x_f)$ on a l'injectivité dans $[0, x_f]$. Comme $x_0 \in [0, x_f]$, on a, en particulier, l'injectivité dans $[x_0, x_f]$. □

On pourrait démontrer une proposition similaire pour le cas infini. Dans le de cas, on peut conclure que la voiture passe par tous les points où la stratégie est valide.

Si c'est clair par le contexte, on dira juste qu'une stratégie est valide, sans faire réference au x en question, pour allegér la notation.

Plus tard on va construire des suites de stratégies. Cependant, on va rémarquer que l'espace métrique des stratégies valides (au sens de la convergence uniforme) n'est pas complet, ce qui nous empêche, d'un point de vue théorique, de trouver des minimums locaux d'une fonction de coût qu'on définira plus tard en construisant des telles suites.

Définition 1.7 (Operateur de validité)**.** *Soit $\mathcal{V}$ l'ensemble des stratégies pour une configuration donnée. Pour $x \in \mathbb{R}_+ \cup \{+\infty\}$, on définit $\mathbb{I}_x : \mathcal{V} \to \{0,1\}$ l'opérateur qui a une stratégie V associe 1 si elle est valide jusqu'à x et 0 si non. On dit que $\mathbb{I}_x$ est l'operateur de validité de la configuration donnée jusqu'à x.*

Proposition 1.2. *Soit $x \in \mathbb{R}_+ \cup \{+\infty\}$. Si, pour la configuration donnée, il existe des stratégies valides et invalides jusqu'à x, $\mathbb{I}_x$ ne dépend pas de façon continue sur $\mathcal{V}$ muni d'une topologie avec plus de deux composantes connexes (en prenant la topologie discrète sur $\{0,1\}$).*

Démonstration. On remarque que $\{0,1\}$ avec la topologie discrète contient deux composantes connexes, puisqu'on trouve que l'union des ouverts disjoints $\{0\}$ et $\{1\}$ est $\{0,1\}$, et que, en tant que singletons, ils sont connexes.

Comme il existe des stratégies valides et invalides, l'application $\mathbb{I}_x$ est surjective. Par conséquent, si c'est une application continue, alors l'ensemble de définition doit contenir la même quantité de composantes connexes que l'ensemble d'arrivée. Or, par hypothèse sur la topologie sur $\mathcal{V}$, ceci n'est pas vrai, et on peut conclure par contraposition que $\mathbb{I}_x$ n'est pas continue. □

En particulier, si on peut trouver des stratégies valides et invalides, l'operateur de validité n'est pas, en général, continu pour la topologie de la convergence uniforme.

L'espace métrique des stratégies défini par $d(V,W) = \sup\big(|V-W|\big)$, où V et W sont des stratégies, restreint aux stratégies invalides n'est pas complet. C'est-à-dire, si on a une suite de stratégies invalides, la limite (du point de vue de la convergence uniforme) peut être valide.

La proposition 1.2 est importante, puisqu'elle a comme conséquence qu'il est très difficile de minimiser des fonctions de coût dépendant de façon continue sur $\mathcal{V}$ pour des stratégies valides.

Intuitivement, on peut comprendre cela de la façon suivante: il peut y avoir une stratégie invalide, laquelle, si on fait un tout petit changement, devient valide. Le choix de la topologie discrète sur $\{0,1\}$ est determiné par le fait qu'on veut distinguer ces deux cas.

Une stratégie V est dite parfaite si et seulement si elle est valide et $\forall t \in \mathbb{R}_+, V(t) > 0$.

Proposition 1.3 (Bijectivité de la primitive)**.** *Soit $S(t) = \int_0^t V(\tau)d\tau$, avec V une stratégie parfaite. On a que $S(t)$ est une bijection continue de $\mathbb{R}_+$ dans $S(\mathbb{R}_+)$.*

Démonstration. Par le théorème fondamental de l'analyse, on a que $S'(t) = V(t)$. On remarque que, par définition $\forall t \in \mathbb{R}_+, S'(t) = V(t) \geq 0$ pour n'importe quelle stratégie. Or, pour les stratégies parfaites, cette inégalite est stricte.

Dans ce cas on remarque que S est strictement monotone et continue. Par le théorème de la bijection continue, on peut conclure que S est une bijection entre $\mathbb{R}_+$ et son image. □

L'intêret de cette proposition se trouve dans la modélisation informatique de cette situation. Si on ne considère que les stratégies parfaites les algorithmes vérifiant des aspects comme la validité des stratégies se simplifient notablement.

On peut représenter la situation graphiques avec des figures telles que la figure 1.

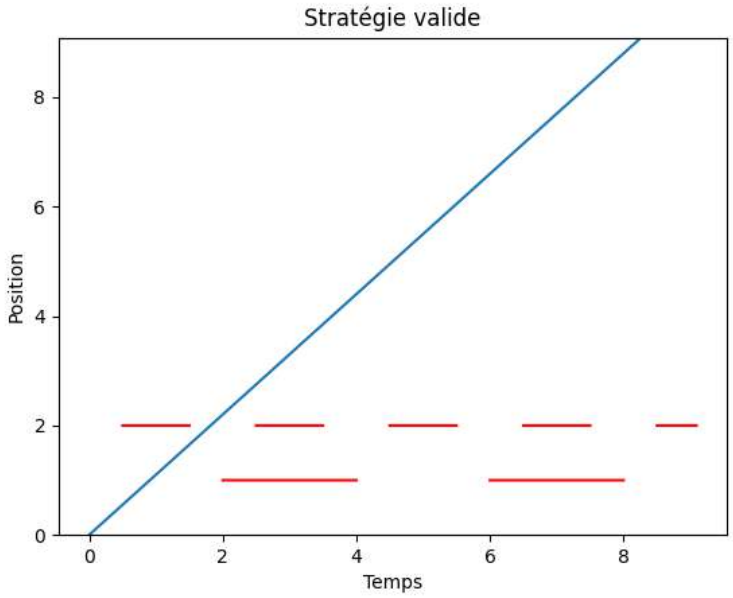


Figure 1: Exemple de figure

2 Stratégies différentiables

La résolution du problème posé dans l'introduction est équivalente à trouver la une fonction qui minimise un coût. Dans cette section on va voir comment définir une telle fonction pour une stratégie valide donnée et on va présenter un algorithme pour la minimiser dans le cas où la stratégie est différentiable.

Définition 2.1 (Sratégie différentiable)**.** *On dit qu'une stratégie V est différentiable si $V \in C^1(\mathbb{R}_+)$ et V' est bornée.*

Cette dernière condition, qui implique que V est continue uniformément, est nécessaire puisqu'elle empêche le cas où la voiture parcourt une distance infinie dans une quantité de temps arbitraire.

On note avec $\mathcal{V}_d$ l'ensemble des stratégies différentiables.

Définition 2.2 (Coût partiel)**.** *Pour les stratégies différentiables on définit le coût partiel, $\hat{\mathcal{C}}$: $\mathcal{V}_d \times \mathbb{R}_+ \to \mathbb{R}$ par $\hat{\mathcal{C}}(V,t) = \int_0^t V(\tau)|V'(\tau)|d\tau$.*

On fait deux remarques:

1. Cette définition n'est que l'intégral de l'énergie cinetique de la voiture a une constante près (la masse de la voiture).

2. La raison pour laquelle on a une notion de coût partiel et que, en fait, on ne veut juste minimiser celui-ci. On veut aussi minimiser le temps totale de parcours.

Parfois c'est intéressant de considerer les fonction $\hat{\mathcal{C}}_{t_0} : \mathcal{V}_d \to \mathbb{R}$ obtenues en fixant un t_0.

Définition 2.3 (Coût total)**.** *Pour un $x \in \mathbb{R}_+$, on définit le coût total $\mathcal{C}_x : \mathcal{V}_d \to \mathbb{R}$, en fonction d'un paramètre $\lambda \in \mathbb{R}_+$ avec $\mathcal{C}_x(V) = \hat{\mathcal{C}}_{\bar{t}}(V) + \lambda \bar{t}$, ou $\bar{t} = \inf(S^{-1}(\{x\}))$, avec $S(t) = \int_0^t V(\tau)d\tau$.*

La définition 2.3 se simplifie dans le cas où on ne considère que les stratégies parfaites (et c'est pour cela que celles-ci sont plus faciles à modéliser).

Algorithme 2.1. *Étant donnée une stratégie différentiable valide V et un $x \in \mathbb{R}_+$, on construit une suite de stratégies de la façon suivante:*

1. *On pose $V_0 = V$ et on choisit ϵ_0.*
2. *$\forall n \in \mathbb{N}^*$, on construit la stratégie U_n, avec $U_n(t) = V_{n-1}(t) + \epsilon_n t$.*
3. *Si la stratégie U_n n'est pas valide, ou si $\mathcal{C}_x(U_n) > \mathcal{C}_x(V_{n-1})$ on ne fait aucun changement sur V (c'est-à-dire, $V_n = V_{n-1}$). Si non, on pose $V_n = U_n$*
4. *On pose $\epsilon_{n+1} = -\epsilon_n/2$.*

Le choix de ϵ_0 détermine la similitude du résultat de l'algorithme et de V.

Proposition 2.1. *La suite obtenue par l'algorithme précedant converge vers une stratégie dont le coût pour le x choisit est inférieur ou égale à l'élément du début.*

Démonstration. Soient V une stratégie différentiable valide et $x \in \mathbb{R}_+$. On applique l'algorithme 2.1 (avec ϵ_0 arbitraire) pour obtenir une suite $(V_n)_{n\in\mathbb{N}}$. On construit une nouvelle suite $S : \mathbb{N} \to \mathbb{R}$, avec $S_n = \mathcal{C}_x(V_n)$ La suite est décroissante par construction. On remarque que S est minorée par 0. C'est-à-dire S est une suite minorée et bornée à valeurs dans une partie non vide de $\mathbb{R}$, puisque $S_0 = \mathcal{C}_x(V) \in S(\mathbb{N})$. On peut conclure que S converge. La décroissance de S nous permet de conclure que $lim_{n\to\infty}(S_n) \leq \mathcal{C}_x(V)$.

On remarque que cette limite est une stratégie différentiable puisqu'elle s'écrit sous la forme $V + \eta t$, avec $\eta \in \mathbb{R}$. □

On appele $\hat{V}_x^{\epsilon_0}$ la stratégie obtenue à partir de V, x et ϵ_0 à partir de l'algorithme 2.1 (qui est une homotopie). On remarque que $\hat{V}_x^{\epsilon_0}(0) = V(0)$ et toutes les derivées d'ordre supérieur à 2 de $\hat{V}_x^{\epsilon_0}$ et de V coïncident. C'est pourquoi on peut dire qu'on a amélioré la stratégie; localement, dans le point $x = 0$, toutes les stratégies de la suite sont égales. Cependant, même si chaque terme de la suite obtenue est une stratégie valide, par la proposition 1.2, on obtient que $\hat{V}_x^{\epsilon_0}$ ne l'est pas forcémment. Néanmoins, on peut trouver des stratégies valides qui lui sont arbitrairement proches (au sens de la convergence uniforme), en prenant des termes de plus en plus grands de la suite.

Dans la figure 2 on trouve un exemple d'optimisation de la stratégie $V(t) = t^2$, avec $\epsilon_0 = 200,\ \lambda = 1,\ N = 100,\ x = 50,\ \varphi_1^x = 10,\ \varphi_1^T = 1.1,\ \varphi_1^\theta = 0,\ \varphi_2^x = 20,\ \varphi_2^T = 2.4$ et $\varphi_2^\theta = 1$. On réduit le coût de 472.33 jusqu'à 408.70.

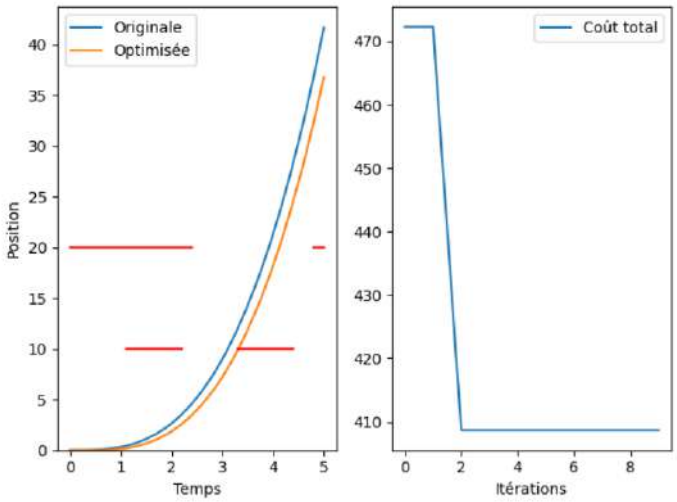


Figure 2: Caption

On finit ce chapitre avec une petite remarque. On peut étendre la définition 2.2 (coût partiel), en permettant que la variable t prenne la valeur $+\infty$, et de la même façon on peut étendre la définition 2.3 (coût total) avec $x = +\infty$. Puisque dans ce qui suit on va parler presque exclusivement des rues finies, la version sans généraliser suffit.

3 Stratégies linéaires

En général ce n'est pas facile de trouver des stratégies optimales (et valides), puisque, comme on a démontré dans la proposition 1.2, la validité ne dépend, en général, de façon continue sur la stratégie. Cependant, malgré cette difficulté il y a des cas où on peut aisément trouver des stratégies optimales.

Définition 3.1 (Stratégie Linéaire Valide)**.** *On appelle Stratégie Linéaire Valide (ou **SLV**) toute stratégie différentiable V qui est valide (jusqu'à un $x \in \mathbb{R}_+ \cup \{+\infty\}$ donné) et qui est une fonction constante.*

C'est-à-dire, V est une **SLV** si et seulement si $\exists v_0 \in \mathbb{R}_+^*, \forall t \in \mathbb{R}_+, V(t) = v_0$.

On note $SLV_x\left[\{\varphi_i\}_{i\in I}\right]$ l'ensemble des **SLV** jusqu'à x d'une configuration $\{\varphi_i\}_{i\in I}$.

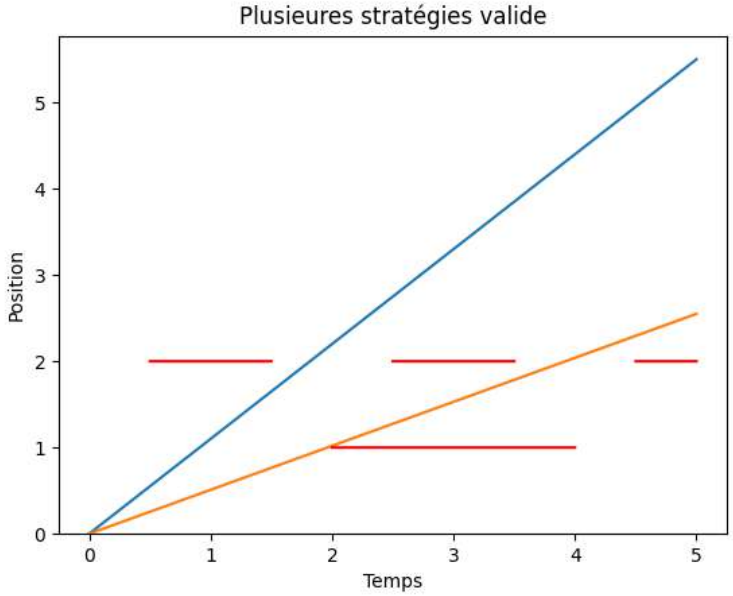


Figure 3: Exemples de **SLV**

Proposition 3.1. *Si on a une configuration qui admet une **SLV**, celle-ci optimise le coût partiel.*

Démonstration. Par définition toute **SLV** V est valide. En plus, on a $\forall t \in \mathbb{R}_+, V'(t) = 0$. Calculons le coût partiel d'une telle stratégie, en posant un $t \in \mathbb{R}_+$ arbitraire:

$$\hat{\mathcal{C}}_t(V) = \int_0^t V(\tau)|V'(\tau)|d\tau = \int_0^t V(\tau)|0|d\tau = 0$$

On remarque que pour n'importe quelle autre stratégie W on a $\hat{\mathcal{C}}_t(W) \geq 0$, puisque $W(\tau)$ et $|W'(\tau)|$ sont positives pour tout $\tau \in [0, t]$ et puisque $t \geq 0$, ce qui nous permet d'appliquer la positivité de l'intégrale.

On a trouvé que $\hat{\mathcal{C}}$ est minoré par 0 et que pour V une **SLV** aribitraire et $t \in \mathbb{R}_+$, on a $\hat{\mathcal{C}}_t(V) = 0$. Donc $\hat{\mathcal{C}}_t(V) = \min_{W \in \mathcal{V}_d} \hat{\mathcal{C}}_t(W)$. C'est-à-dire, V optimisie le coût partiel. □

Pour le cas du coût totale le résultat n'est pas si bon. Cependant, on peut le majorer avec ce qu'on a trouvé. Pour cela, on a besoin du résultat suivant:

Proposition 3.2. *Soient $\{\varphi_i\}_{i\in I}$ une configuration et $x \in \mathbb{R}_+$. On a que $\forall V \in SLV_x\left[\{\varphi_i\}_{i\in I}\right], C_x = \lambda\frac{x}{v_0}$. Ici, $\lambda \in \mathbb{R}_+$ est le paramètre défini pour le coût total et $v_0 \in \mathbb{R}_+$ est l'unique valeur telle que $\forall t \in \mathbb{R}_+, V(t) = v_0$.*

Démonstration. Soit V une telle stratégie. On a montré dans la proposition 3.1 que le coût partiel de toute **SLV** jusqu'à n'importe quel x est 0. Donc, pour V, on a

$$C_x(V) = \hat{C}_{\bar{t}}(V) + \lambda\bar{t} = \lambda\bar{t} \tag{1}$$

On rappelle que $\bar{t} = \inf\big(S^{-1}(\{x\})\big)$, avec $S(t) = \int_0^t V(\tau)d\tau$. Ici, $S(t) = v_0 t$ (en faisant l'hypothèse $S(0) = 0$, ce qui simplifie les calculs mais ne change pas le résultat). Donc, par bijectivité de S, on a $S^{-1}(\{x\})$ est un singleton, et alors $\bar{t} = S^{-1}(x)$. Explicitement, $\bar{t} = \frac{x}{v_0}$. On substitue dans (1), pour obtenir $C_x(V) = \lambda\frac{x}{v_0}$.

□

Proposition 3.3. *Soit $\{\varphi_i\}_{i\in I}$ une configuration qui admet des **SLV**. On a pour chaque $x \in \mathbb{R}_+$, $\inf\limits_{W\in\mathcal{V}_d} \mathcal{C}_x(W) \leq \lambda\frac{x}{v_0}$, où, pour $t \in \mathbb{R}_+$ arbitraire, $v_0 = \sup\limits_{V\in SLV_x[\{\varphi_i\}_{i\in I}]} V(t)$.*

Démonstration. Soient $\{\varphi_i\}_{i\in I}$ une configuration et $x \in \mathbb{R}_+$ qui vérifient les hypothèses de la proposition. Prenons une stratégie $W \in \mathcal{V}_d$, dont le coût total est supérieur à $\lambda\frac{x}{v_0}$. Montrons qu'il existe une **SLV** dont le coût est infèrieur à celui de W.

On pose $v' \in \mathbb{R}_+$ tel que $C_x(W) = \lambda\frac{x}{v'}$ et on remarque que $v' < v_0$. Par construction de v_0, $\exists v \in \mathbb{R}_+$, tel que $v_0 \geq v > v'$ et tel qu'il y a $V \in SLV_x\big[\{\varphi_i\}_{i\in I}\big]$ avec $\forall t \in \mathbb{R}_+, V(t) = v$.

Donc, par la proposition 3.2, on a

$$C_x(W) = \lambda\frac{x}{v'} > \lambda\frac{x}{v} = C_x(V)$$

On a trouvé que, pour n'importe quelle stratégie W qui a un coût total supérieur à $\lambda\frac{x}{v_0}$ (avec v_0 défini comme dans l'énoncé) il existe une autre stratégie V avec $C_x(V) < C_x(W)$. On peut conclure que $\inf\limits_{W\in\mathcal{V}_d} \mathcal{C}_x(W) \leq \lambda\frac{x}{v_0}$.

□

Voyons quelles conditions doit vérifier une configuration $\{\varphi_i\}_{i\in I}$ pour qu'il y ait des **SLV**.

L'idée génerale va être de construire un ensemble, qu'on va appeller fenêtre de validité, qui va nous donner pour chaque configuration les valeurs $v_0 \in \mathbb{R}_+$ pour lesquelles on peut construire des **SLV**.

Définition 3.2 (Fenêtre de validité)**.** *Soit $\{\varphi_i\}_{i\in I}$ une configuration. On défini l'ensemble $\mathcal{F}_x[\{\varphi_i\}_{i\in I}]$ par*

$$\mathcal{F}_x[\{\varphi_i\}_{i\in I}] = \big\{v_0 \in \mathbb{R}_+ \mid \big(V(t) \in \mathcal{V}_d, t \mapsto v_0\big) \in SLV_x\big[\{\varphi_i\}_{i\in I}\big]\big\}$$

Si la configuration est claire par le contexte on peut écrire $\mathcal{F}_x$ pour la fenêtre de validité. Le prochain objectif sera de trouver cet ensemble pour une configuration donnée,

Tout d'abord on va considerer une question un peu plus simple. Étant donnée une configuration constitué par un seul feu, quelle est la valeur de $\mathcal{F}_x$?

Proposition 3.4. *Soit $\{\varphi_i\}_{i\in\{i\}}$ une configuration avec un seul feu et $x > \varphi_i^x$. On a:*

$$\mathcal{F}_x[\{\varphi_i\}_{i\in\{i\}}] = \mathbb{R}_+^* \cap \left(\bigcup_{k\in\mathbb{Z}}\right]\frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T}\Big]\Big)$$

Démonstration. Soit v telle que $V(t) = v$ et $V \in SLV_x[\{\varphi_i\}_{i\in I}]$. Puisque $v > 0$, la définition de validité garantie qu'une stratégie linéaire de la forme $V(t) = v$ est valide pour $\{\varphi_i\}_{i\in\{i\}}$ si et seulement si v est une solution du système suivant:

$$\begin{cases} v = \frac{\varphi_i^x}{t} \\ \lfloor \frac{t}{\varphi_i^T} + \varphi_i^\theta \rfloor \equiv 1(mod\ 2) \\ v > 0 \end{cases} \tag{1}$$

On écrit $t = \frac{\varphi_i^x}{v}$ et on substitue dans la deuxième équation, pour obtenir

$$\lfloor \frac{\varphi_i^x}{v\varphi_i^T} + \varphi_i^\theta \rfloor \equiv 1(mod\ 2)$$

Pour chaque $k \in \mathbb{Z}$, on obtient des différents valeurs possibles de v vérifiant l'équation suivante:

$$\lfloor \frac{\varphi_i^x}{v\varphi_i^T} + \varphi_i^\theta \rfloor = 2k + 1$$

Fixons un tel k. On s'intéresse à l'intervalle suivant:

$$2k + 1 \leq \frac{\varphi_i^x}{v\varphi_i^T} + \varphi_i^\theta < (2k + 1) + 1 \tag{2}$$

Par la première inégalité, on obitent (en employant le fait que chaque terme à droite est positif), pour $2k + 1 - \varphi_i^\theta > 0$:

$$v \leq \frac{\varphi_i^x}{(2k + 1 - \varphi_i^\theta)\varphi_i^T}$$

Pour $2k + 1 - \varphi_i^\theta < 0$.

$$v \geq \frac{\varphi_i^x}{(2k + 1 - \varphi_i^\theta)\varphi_i^T}$$

Or, ce dernier cas n'est pas intéressant, puisque, en utilisant le fait que $v > 0$ (troisième equation de (1)) et que φ_i^x et φ_i^T sont positifs, on a:

$$v > 0 \geq \frac{\varphi_i^x}{(2k + 1 - \varphi_i^\theta)\varphi_i^T}$$

Donc, on ne s'intéresse qu'au premier cas (et on fait la remarque qu'on devra prendre une intersection avec $\mathbb{R}_+^*$ plus tard).

Maintenant on va prendre la deuxième inégalité de (2). On obtient, si $2k + 2 - \varphi_i^\theta > 0$

$$v > \frac{\varphi_i^x}{(2k + 2 - \varphi_i^\theta)\varphi_i^T}$$

Et, si $2k + 2 - \varphi_i^\theta < 0$

$$v < \frac{\varphi_i^x}{(2k + 2 - \varphi_i^\theta)\varphi_i^T}$$

Or, par le même argument qu'avant, on ne s'intéresse pas a ce cas. Donc, en combinant les résultats précedants, on trouve que l'ensemble des solutions de (1) (ce qui se correspond à la fenêtre de validité de la configuration) est:

$$\mathbb{R}_+^* \cap \left(\bigcup_{k\in\mathbb{Z}} \left] \frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T} \right] \right)$$

□

Maintenant, on va construire la fenêtre de validité pour une configuration en général. Ici $\#I$ dénote la cardinalité de l'ensemble I.

Proposition 3.5. *Soit $\{\varphi_i\}_{i\in I}$ une configuration. On construit $\#I$ nouvelles configurations, chacune d'entre elles étant juste des singletons de feux. On a que:*

$$\mathcal{F}_x[\{\varphi_i\}_{i\in I}] = \bigcap_{i\in I} \mathcal{F}_x[\{\varphi_i\}_{i\in\{i\}}]$$

Démonstration. On remarque que $\mathcal{F}_x[\{\varphi_i\}_{i\in I}]$ est l'ensemble des $v \in \mathbb{R}_+^*$ qui satisfont le systeme:

$$\bigwedge_{i\in I} \left(v = \frac{\varphi_i^x}{t_i} \right) \wedge \left(\lfloor \frac{t_i}{\varphi_i^T} + \varphi_i^\theta \rfloor \equiv 1 (mod 2) \right)$$

Ceci suit directement des définitions de stratégie valide et de stratégie linéaire.

On remarque que ce n'est que la conjonction des conditions qui caractérisent le fait d'appartenir à la fenêtre de validité de chaque configuration singleton, par la proposition 3.4. Donc, l'ensemble de solutions de ce systeme, $\mathcal{F}_x[\{\varphi_i\}_{i\in I}]$, et l'union des ensembles de solutions corréspondants à ces configurations, ce qui achève la preuve. □

Grâce à cette dernière proposition, on peut dire que, pour une configuration $\{\varphi_i\}_{i\in I}$ arbitraire et un $x \in \mathbb{R}_+$:

$$\mathcal{F}_x[\{\varphi_i\}_{i\in I}] = \bigcap_{i\in I} \left(\bigcup_{k\in\mathbb{Z}} \left] \frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T} \right] \cap \mathbb{R}_+^* \right)$$

Tout ceci nous permet de conclure avec le théorème suivant.

Théorème 3.1 (Existence des **SLV**)**.** *Soit $x \in \mathbb{R}_+$. Une configuration $\{\varphi_i\}_{i\in I}$ admet des **SLV**, jusqu'à x si et seulement si*

$$\bigcap_{i\in I} \left(\bigcup_{k\in\mathbb{Z}} \left] \frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T} \right] \cap \mathbb{R}_+^* \right) \neq \emptyset$$

Démonstration. Un système n'admet pas de **SLV** s'il n'y a pas de $v \in \mathbb{R}_+$ tel que la stratégie $V(t) = v$ soit valide. On peut conclure la démonstration en utilisant la définition de fenêtre de validité et les propositions 3.4 et 3.5.

□

Proposition 3.6. *Pour n'importe quelle configuration $\{\varphi_i\}_{i\in I}$, et $x \in \mathbb{R}_+$ l'ensemble F_x est, soit vide, soit infini.*

Démonstration. On prend la configuration donnée par $\{\varphi_i\}_{i\in I}$, avec $I = \{1, 2\}$ ...
Supposons que F_x n'est pas vide. Alors, on choisit un $v \in F_x$, qui doit vérifier que $\exists k \in \mathbb{Z}$, et $2k + 1 - \varphi_i^\theta > 0$. Pour ce choix, on a $\forall i \in I$,

$$v \in \left]\frac{\varphi_i^x}{(2k+2-\varphi_i^\theta)\varphi_i^T}, \frac{\varphi_i^x}{(2k+1-\varphi_i^\theta)\varphi_i^T}\right]$$

□

Pour le cas où $\mathcal{F}_x = \emptyset$ on pourra quand-même décomposer la configuration en plusieures parties, où chacune d'entre elles admet des **SLV**. Dans ce cas, on pourra construir une composition de **SLV**, en changeant de vitesse chaque fois qu'on en aura besoin. Évidemmenet la stratégie trouvé ainsi ne sera pas différentiable.

L'utilité du théorème 3.1 est que, avant de construir une rue et de configurer ses feux, on a une méthode pour savoir s'il y a des vitesses auxquelles on peut traverser la rue. De façon analogue, étant donnée une vitesse maximale on peut nous assurer que la configuration de feux que l'on va mettre en place permet de la traverser à cette vitesse-ci.

4 Configurations et états

Dans cette partie on va voir plus en détail quelques résultats que l'on peut obtenir à partir de nos définitions de feux, configurations et états. Ici, on va utiliser le fait que la periode en vert égale la période en rouge. On pourrait aussi démontrer des résultats similaires dans le cas plus général.

Proposition 4.1. *Soit $\{\varphi_i\}_{i\in I}$ une configuration finie. Si $\forall i, j \in I, \frac{\varphi_i^T}{\varphi_j^T} \in \mathbb{Q}$, $\mathcal{E}(\{\varphi_i\}_{i\in I}, t)$ est périodique.*

Démonstration. On remarque que pour $i, j \in I$,

$$\left(\frac{\varphi_i^T}{\varphi_j^T} \in \mathbb{Q}\right) \iff \left(\exists q_{ij}, p_{ij} \in \mathbb{N} \mid \frac{\varphi_i^T}{\varphi_j^T} = \frac{q_{ij}}{p_{ij}}\right) \tag{1}$$

Si on inverse les rôles de i et j on obtient $p_{ij}\varphi_j^T = q_{ji}\varphi_i^T$. C'est-à-dire, $p_{ij} = q_{ji}$, ce qui nous permet d'écrire (1) sous la forme $\frac{\varphi_i^T}{\varphi_j^T} = \frac{q_{ij}}{q_{ji}}$.

Soit $h \in I$ tel que $\varphi_h^T = \min(\{\varphi_i^T\}_{i\in I})$. On pose

$$\Psi = 2\prod_{i,j\in I} q_{ij}\varphi_h^T$$

On va montrer que $\forall k \in I$, φ_k^f est périodique de période Ψ (et donc, $\mathcal{E}(\{\varphi_i\}_{i\in I}, t)$ l'est aussi). Pour ceci, on va montrer que $\varphi_k^f(t) = 0 \iff \varphi_k^f(t + \Psi) = 0$ et que $\varphi_k^f(t) = 1 \iff \varphi_k^f(t + \Psi) = 1$. On ne va expliciter que le premier cas, puisque la démonstration est pareille pour l'autre.

$$\varphi_k^f(t + \Psi) = 0 \iff \lfloor\frac{t + \Psi}{\varphi_k^T} + \varphi_k^\theta\rfloor \equiv 0 \ (mod\ 2) \tag{2}$$

Si on substitue, on obtient:

$$\frac{\Psi}{\varphi_k^T} \equiv 0 \ (mod\ 2) \iff \frac{\prod\limits_{i,j\in I} q_{ij}\varphi_h^T}{\varphi_k^T} \in \mathbb{N} \tag{3}$$

Montrons que:

$$\frac{\prod\limits_{i,j\in I} q_{ij}\varphi_h^T}{\varphi_k^T} \in \mathbb{N}$$

On remarque que $\frac{\varphi_h^T}{\varphi_k^T} = \frac{q_{hk}}{q_{kh}}$. Donc

$$\frac{\prod\limits_{i,j\in I} q_{ij}\varphi_h^T}{\varphi_k^T} = \frac{\prod\limits_{i,j\in I} q_{ij}q_{hk}}{q_{kh}}$$

On a que q_{kh} est un facteur du nominateur, alors on peut factoriser. Le résultat est un produit d'entiers, qui, par stabilité du produit en $\mathbb{N}$ est un entier aussi. Ainsi, par (3) on a démontré que $\frac{\Psi}{\varphi_k^T} \equiv 0 \ (mod\ 2)$, ce qui nous permet de déduire

$$\varphi_k^f(t+\Psi) = 0 \iff \lfloor \frac{t+\Psi}{\varphi_k^T} + \varphi_k^\theta \rfloor \equiv 0 \ (mod\ 2) \iff \lfloor \frac{t}{\varphi^k} + \varphi_k^\theta \rfloor \equiv 0 \ (mod\ 2) \iff \varphi_k^f(t) = 0$$

□

On remarque que la periodicité est vrai dans le cas fini. On obtient, donc, un résultat intéressant. Pour le cas des configurations inifinies, on peut trouver que l'état de chaque partie finie soit périodique, mais que tout l'état le soit.

Proposition 4.2. *Soit $\{\varphi_i\}_{i\in I}$ une configuration finie. $\forall \varepsilon > 0, \exists \{\varphi'_i\}_{i\in I}$ une autre configuration, telle que $\forall \varphi_j \in \{\varphi_i\}_{i\in I}, |\varphi_j^T - (\varphi')_j^T| < \varepsilon$ et telle que $\mathcal{E}(\{\varphi'_i\}_{i\in I}, t)$ est périodique.*

Démonstration. Soit $\varepsilon > 0$. Choisissons $\varphi_h \in \{\varphi_i\}_{i\in I}$ tel que $\varphi_h^T = \min(\{\varphi_i^T\}_{i\in I})$. Pour chaque élément $\varphi_j \in \{\varphi_i\}_{i\in I}$ on pose φ'_j avec $(\varphi')_j^\theta = \varphi_j^{\theta}$‘, $(\varphi')_j^x = \varphi_j^x$ et $(\varphi')_j^T = q_j\varphi_h^T$, tel que $(\varphi')_j^T \in]\varphi_j^T - \varepsilon, \varphi_j^T + \varepsilon[$. Pour justifier qu'il existe un tel q_j, on remarque qu'on a besoin de $q_j \in]\frac{\varphi_j^T - \varepsilon}{\varphi_h^T}, \frac{\varphi_j^T + \varepsilon}{\varphi_h^T}[$. Comme c'est un inteervale (non-vide) de $\mathbb{R}$ on conclut par la densité de $\mathbb{Q}$ dans $\mathbb{R}$.

La nouvelle configuration $\{\varphi'_i\}_{i\in I}$ possède un état périodique puisque tous ses termes ont un ratio rationel deux à deux, et donc on peut appliquer la proposition 4.1.

□

En conclusion, on peut approcher n'importe quelle partie finie d'une configuration à une configuration périodique.

5 Moyennes et choix de configurations

Jusqu'à l'instant, on a vu la situation principalement du point de vue du conducteur, qui, face à une configuration de feux donnée, doit choisir une stratégie pour minimiser le coût énergique.

Maintenant on va se fixer sur le point de vue d'un architecte, pour, entre autres choses, choisir la meilleur position, période et phase des feux.

On remarque que, pour ce fait, on ne va plus considerer qu'une seule voiture, mais "toutes les voitures possibles" en même temps. En plus, on va fixer deux nouvelles contraîntes. Dans la section qui suit toute configuration $\{\varphi_i\}_{i\in I}$ doit admettre un état périodique $\mathcal{E}$, de période Ψ, et il doit y avoir une vitesse maximale permise, v_{max}.

La deuxième est cohérente d'un point de vue phisique, et la première et extrémment utile dans ce qui suit. En plus, grâce à la proposition 4.1 on sait que ce n'est pas une vrai constrainte, dans le sens où pour n'importe quelle partie finie de n'importe quelle configuration on peut l'approcher par une telle configuration périodique.

En plus on va définir un nouveau type de stratégie; qui est en fait celle que la plupart des conducteurs choisissent de façon naturelle.

1. Si possible, $V(t) = v_{max}$.
2. S'il y a un feu, $V(t) = 0$.

On appelle cette type de stratégie une **stratégie naturelle**. De façon plus précise, on les définit comme:

Définition 5.1 (Stratégie Naturelle)**.** *Une stratégie est dit naturelle si*

On remarque que, pour chaque partie où la vitesse n'est pas nulle on a une **SLV**. On dénote avec $\mathcal{N}$ l'ensemble des stratégies naturelles pour une configuration.

Définition 5.2 (Coût naturel moyen)**.** *On définit le coût naturel moyen d'une stratégie par:*

$$\mu_x(\{\varphi_i\}_{i\in I}, v_{max}) = \frac{1}{\Psi}\int_0^{\Psi} C_x\big(N_{v_{max}}(t_0)\big)dt_0$$

On rapelle que Ψ est la période de la configuration donnée.

On remarque que ceci se correspond au coût moyen des stratégies naturelles pour une configuration jusqu'à un point x. En pratique c'est très difficile d'utiliser cette définition. Or, on peut l'approcher avec des sommes de Riemmann.

On va maintenant répondre à la question suivante:

On a une longue rue intersectée par plein de rues sécondaires. On connait les positions des carrefours et les périodes de chaque feu. Quelles phases est-ce que l'on devrait choisir pour les feux pour que les voitures qui passent par la rue consomment, en moyenne, le moins d'énergie possible?

On a developpé les outils qui nous permettent de poser cette question de forme précise. Étant donnée une configuration finie de feux $\{\varphi_i\}_{i\in I}$ et connaisant pour chaque φ_i sa position et sa période, on cherche à choisir les φ_i^θ qui minimisent $\mu_x[(\{\varphi_i\}_{i\in I}, v_{max})]$ pour un x et un v_{max} donnés.

On trie la configuration en ordre croissant des positions des feux (on rapelle que deux feux ne peuvent pas partager position, par définition), et on elimine les feux dont la position et supérieur

où égale à x. On obtient donc $(\varphi_i)_{i\in I'}$, avec $I' \subseteq I$; la famille ordonnée.

Pour des raisons de lisibilité on va écrire l'algorithme en forme de Pseudocode et on va prendre la convention suivante: comme la nouvelle configuration est ordonnée, on se permitera d'écrire $i+1$ pour dénoter l'indice se correspondant au suivant élément de celui qui possede indice i.

La variable M va dénoter la configuration qui à un instant donné a eu la meilleure moyenne.

Algorithme 5.1. *On prend $i \in I'$ tel que $\varphi_i = \min_{j\in I'}(\varphi_j^x)$ et on lui applique la méthode suivante:*

***Algorithme**(i)*

Require: $N > 0$
  $\varphi_i^\theta \leftarrow \varphi_i^\theta + \frac{\varphi_i^T}{N}$
  if $\mu_x[\{\varphi_j\}_{j\in I'}, v_{max}] < \mu_x[M, v_{max}]$ ***then***
    $M \leftarrow \{\varphi_j\}_{j\in I'}$
  end if
  if $\varphi_i^\theta = \varphi_i^T$ ***then***
    if $\varphi_i^x \neq \max(\{\varphi_j^x\}_{j\in I'})$ ***then***
      ***Algorithme**($i+1$)*
    end if
    $\varphi_i^\theta \leftarrow 0$
    return
  else if $\varphi_i^\theta \neq \varphi_i^T$ ***then***
    ***Algorithme**($i+1$)*
  end if

Conjecture 5.1. *Lorsque $N \to +\infty$, la suite construite en prenant par chaque terme le résultat de l'algorithme 5.1 appliqué à N converge vers la configuration qui minimise μ_x, étant données les positions et les périodes des feux.*

Conjecture 5.2. *La complexité de l'algorithme 5.1 est $O(n!)$ par rapport au nombre de feux dans la configuration (si on fixe tout le reste des paramètres).*

6 Discontinuité de la fonction de coût

Dans le vrai monde on remarque que les feux peuvent être mal configurés accidentalement. C'est-à-dire, il peut y avoir des petits variations dans la phase de chaque feu. *A priori* cela ne semble pas être trop problematique. Or, on remarque que le coût moyen de la configuration n'est pas une fonction continue en prenant comme variables les phases.

Ce résultat a été confirmé experimentalement par le modèle (en particulier l'algorithme 5.1), et est simple à justifier. Ici, le coût va être défini comme proportionnel au nombre de fois que la voiture accélere. En effet, si on bouge le deuxième feu de la figure 4, on remarque que toutes les voitures qui ne devaient faire qu'un seul arret doivent maintenant en faire deux, augmentant le coût moyen.

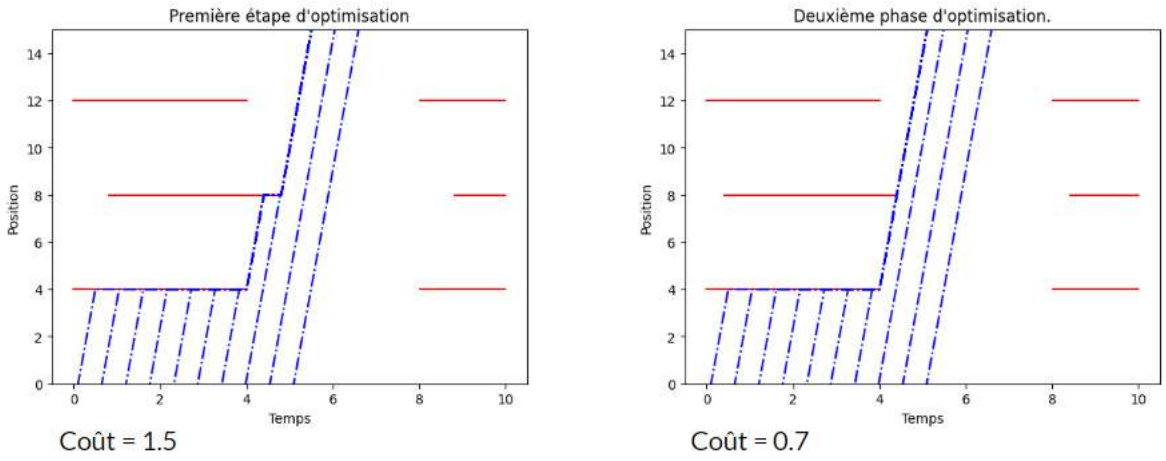


Figure 4: Exemple d'optimisation d'une configuration

7 Modèle

Dans cette section on va joindre le code que l'on peut utiliser pour, entre autres choses vérifier la validité d'une stratégie, pour l'optimiser si c'est le cas et pour définir des configurations sujetes à certaines contraîntes.

On peut voir comment on a traduit les définitions mathématiques en objets informatiques avec lesquels on peut faire des calculs.

```python
import numpy as np
from scipy.integrate import quad
from scipy.misc import derivative
import matplotlib.pyplot as plt
import math
from typing import List

class Feu():
    '''
    Un feu, qui est un   lment   d'une configuration.

    Properties
    ----------
    position : float
        Position que le feu occupe dans la rue. R el positif.
    periode : float
        P riode du feu. Elle est la m me pour l' tat  vert et pour le rouge.
    phase : float
        Phase initial du feu.    valeurs entre 0 et la p riode.
    '''
    def __init__(self, position : float, periode : float, phase : float) -> None:
        self.position = position
        self.periode = periode
        self.phase = phase

    @property
    def position(self) -> float:
        '''
        Getter pour la position.

        Returns
        -------
        float
            Position du feu.
        '''
        return self.__position

    @position.setter
    def position(self, pos) -> None:
        '''
        Setter pour la position.

        Raises
        ------
        ValueError
            Si la position est n gative.
        '''
        if pos<0:
            raise ValueError("La position doit  tre  positive.")
        self.__position = pos

    @property
    def periode(self) -> float:
        '''
        Getter pour la p riode.

        Returns
        -------
        float
            Periode du feu
```

```
        '''
        return self.__periode

    @periode.setter
    def periode(self, per) -> None:
        '''
        Setter pour la p riode.

        Raises
        ------
        ValueError
            Si la p riode est n gative.
        '''
        if per<0:
            raise ValueError("La p riode doit  tre  positive.")
        self.__periode = per

    @property
    def phase(self) -> float:
        '''
        Getter pour la phase.

        Returns
        -------
        float
            Phase du feu.
        '''
        return self.__phase

    @phase.setter
    def phase(self, ph) -> None:
        '''
        Setter pour la phase.

        Raises
        ------
        ValueError
            Si la phase n'est pas entre 0 et la p riode.
        '''
        if ph<-self.__periode or ph>self.__periode:
            raise ValueError("La phase doit  tre  entre -p riode et la p riode.")
        self.__phase = ph


    def fonction(self, t : float) -> int:
        '''
        M thode qui indique si le feu est en vert ou en rouge.

        Parameters
        ----------
        t : float
            R el positif. Temps pour lequel on veut conna tre l' tat  du feu.

        Returns
        -------
        0
            Si le feu est en rouge.
        1
            Si le feu est en vert.
        '''
```

```
        return int(t/self.periode+self.phase)%2

    def fonction_graphe(self, t) -> None | int:
        '''
        Version de la fonction du feu utile pour dessiner les graphes.

        Parameters
        ----------
        t : float
            R el positif. Temps pour lequel on veut conna tre l' tat  du feu.

        Returns
        -------
        None
            Si le feu est en rouge.
        float
            Position du feu s'il est en vert.
        '''
        if self.fonction(t)==0:
            return None
        return self.position


class Configuration():
    '''
    Liste de feux.

    Properties
    ----------
    feux : list of Feu
        Feux qui constituent la configuration.
    '''
    def __init__(self, liste_feux : List[Feu]) -> None:
        self.feux = liste_feux

    @property
    def feux(self) -> List[Feu]:
        '''
        Getter pour la liste de feux.

        Returns
        -------
        list of Feu
            Liste des feux.
        '''
        return self.__feux

    @feux.setter
    def feux(self, feux):
        '''
        Setter pour la liste de feux.

        Raises
        ------
        TypeError
            Si la liste n'est pas constitu e de feux.
        ValueError
            S'il y a deux feux dans la m me position.
        '''
        for feu in feux:
```

```
            if not(isinstance(feu,Feu)):
                raise TypeError("La configuration doit  tre  une liste de feux.")

        positions = [feu.position for feu in feux]
        if len(list(set(positions)))!=len(positions):
            raise ValueError("Il y a deux feux    la m me position.")

        self.__feux = feux

    def periode(self) -> float:
        '''
        Calcule la p riode de la configuration (ou une approximation si celle-ci n'
                                                existe pas).
        '''
        min_periode = min([f.periode for f in self.feux])
        psi = 2*min_periode
        for i in range(len(self.feux)):
            for j in range(len(self.feux)):
                psi = psi*self.feux[i].periode/self.feux[j].periode
        return psi

    def position_naturelle(self, t_max : float, v_max : float, t_0=0, N=100000,
                                              epsilon=1e-5) -> np.array:
        '''
        Construit la position de la strat gie naturelle qui commence    un temps
                                                initial donn .

        Parameters
        ----------
        t_max : float
            Temps maximal pour lequel on veut conna tre la position de la voiture.
        v_max : float
            Vitesse maximale permise dans la rue.
        t_0 : float
            Temps auquel la voiture arrive au d but de la rue (default: 0).
        N : int
            Nombre d' taps  pour le calcul (default: 100000).
        epsilon : float
            Facteur de pr cision (default 1e-5).

        Returns
        -------
        np.array
            Positions associ e    la strat gie naturelle commen ant en ce moment.
        '''
        temps = np.linspace(0, t_max, N)
        pos = np.zeros(N)-1
        coll = False
        for i, t in enumerate(temps):
            if t>t_0:
                break
        for n in range(i, N):
            for feu in self.feux:
                if feu.fonction_graphe(temps[n-1])!=None:
                    if abs(feu.fonction_graphe(temps[n-1])-pos[n-1])<=epsilon:
                        pos[n] = pos[n-1]
                        coll = True
                        break
                    else:
                        coll = False
```

```
                else:
                    coll = False
            if not(coll):
                pos[n] = pos[n-1]+(t_max/N)*v_max
        return pos

    def strategie_naturelle(self, t : float, v_max : float, t_0=0, N=10000, epsilon=
                                                1e-5) -> callable:
        '''
        Construit la strat gie naturelle qui commence    un temps initial donn .

        Parameters
        ----------
        t : float
            Temps pour lequel on veut conna tre la vitesse de la voiture.
        v_max : float
            Vitesse maximale permise dans la rue.
        t_0 : float
            Temps auquel la voiture arrive au d but de la rue (default: 0).
        N : int
            Nombre d' taps  pour le calcul (default: 10000).
        epsilon : float
            Facteur de pr cision (default 1e-5).

        Returns
        -------
        Strategie
            Strat gie naturelle commen ant en ce moment.
        '''
        fonc = lambda s: self.position_naturelle(s, v_max, t_0, N, epsilon)
        raise NotImplementedError()
        return Strategie(derivative(fonc, t, dx=1e-9))

    def cout_discret(self, positions : list) -> int:
        '''
         value  le co t discret d'une strat gie naturelle.

        Parameters
        ----------
        positions : list
            Positions de la strat gie naturelle.

        Returns
        -------
        int
            Quantit  de changements de vitesse.
        '''

        compte = 0
        for i in range(len(positions)-2):
            if positions[i+1] == positions[i] and positions[i+2] != positions[i] and
                                                    positions[i] != -1:
                compte += 1
        return compte

    def cout_moyen(self, t_max : float, t_debut_max : float, v_max : float, N : int)
                                                -> float:
        '''
        Calcule le co t moyen (changements) des strat gies naturelles jusqu'   un
                                                    point donn .
```

```
        Parameters
        ----------
        t_max : float
            Temps total    calculer.
        t_debut_max : float
            Temps maximal pour lequel on veut conna tre le co t moyen.
        v_max : float
            Vitesse maximale permise.
        x : float
            Position pour laquelle on va calculer le co t moyen.
        N : int
            Taille d'  chantillonage  .

        Returns
        -------
        float
            Co t moyen des strat gies naturelles pour la configuration.
        '''
        moyenne = 0
        for t_0 in np.linspace(0, t_debut_max, N):
            positions = self.position_naturelle(t_max, v_max, t_0=t_0)
            moyenne += self.cout_discret(positions)
        return moyenne/N

    def reduit_cout_moyen(self, t_max : float, t_debut_max : float, v_max : float, N
                                                    : int) -> list:
        '''
        Calcule le meilleures phases pour que le co t moyen soit minimal.

        Parameters
        ----------
        t_max : float
            Temps total    calculer.
        t_debut_max : float
            Temps maximal pour lequel on veut conna tre le co t moyen.
        v_max : float
            Vitesse maximale permise.
        x : float
            Position pour laquelle on va calculer le co t moyen.
        N : int
            Taille d'  chantillonage  .

        Returns
        -------
        list
            Liste avec les meilleurs phases.
        '''
        feux = self.feux.copy()
        global M
        M = [feu.phase for feu in self.feux]

        def algorithme(i):
            feux[i].phase = feux[i].phase + feux[i].periode/N
            conf2 = Configuration(feux)
            if conf2.cout_moyen(t_max, t_debut_max, v_max, N) <= self.cout_moyen(
                                                        t_max, t_debut_max, v_max, N):
                M = [feu.phase for feu in self.feux]

            if feux[i].phase==feux[i].periode:
```

```
                if i != len(feux)-2:
                    algorithme(i+1, M, feux)
                feux[i].theta = 0
                return
            else:
                if i!= len(feux)-1:
                    algorithme(i+1)

        algorithme(0)
        return M


class Strategie():
    '''
    Fonction vitesse choisie par le conducteur de la voiture.

    Attributes
    ----------
    vitesse : callable
        Profil de vitesse.
    '''
    def __init__(self, fonction : callable) -> None:
        self.vitesse = fonction

    def est_valide(self, configuration : Configuration, x : float, N=10000) -> bool:
        '''
        V rifie si la strat gie est valide pour une configuration donn e jusqu'
                                                un point.

        Parameters
        ----------
        configuration : Configuration
            Configuration pour laquelle on veut v rifier la validit .
        x : float
            Position jusqu'   laquelle on veut savoir si la strat gie est valide.
        N: int
            Pr cision pour trouver la pr image (default: 10000).

        Returns
        -------
        bool
            true si la configuration est valide, false sinon.
        '''
        t_max=0
        while self.position(t_max)<x:
            t_max += 1/N
        t_tilde = np.linspace(0, t_max, N)
        for t in t_tilde:
            for feu in configuration.feux:
                if(math.isclose(feu.position, self.position(t), abs_tol=1e-2)):
                    if self.vitesse(t)!=0 and feu.fonction(t) == 1:
                        return False
        return True

    def position(self, t : float) -> float:
        '''
        Calcule num riquement l'int grale de la vitesse.

        Parameters
        ----------
```

```
        t : float
            Temps pour lequel on veut conna tre la position.

        Returns
        -------
        float
            Position du feu au temps donn .

        Raises
        ------
        ValueError
            Si le temps est n gatif.
        '''
        if t<0:
            raise ValueError("Le temps doit  tre  positif.")
        return quad(self.vitesse, 0, t)[0]

    def acceleration(self, t : float) -> float:
        '''
        Calcule num riquement la deriv e de la vitesse.

        Parameters
        ----------
        t : float
            Temps pour lequel on veut conna tre l'acceleration.

        Returns
        -------
        float
            Acceleration du feu au temps donn .

        Raises
        ------
        ValueError
            Si le temps est n gatif.
        '''
        if t<0:
            raise ValueError("La position doit  tre  positive.")
        return derivative(self.vitesse, t, dx=1e-9)

    def evalue_cout_partiel(self, t : float) -> float:
        '''
        Calcule le co t partiel (version diff rentiable) de la strat gie.

        Parameters
        ----------
        t : float
            Temps pour lequel on veut calculer le co t partiel.

        Returns
        -------
        float
            Co t partiel jusqu'au temps donn .
        '''
        return quad(lambda x: self.vitesse(x)*abs(self.acceleration(x)), 0, t)[0]

    def evalue_cout_total(self, x : float, lam : float, epsilon=1e-5, N=200) -> float
                                                    :
        '''
        Calcule le co t total    une position donn e.
```

```
        Parameters
        ----------
        x : float
            Position pour laquelle on va calculer le co t.
        lam : float
            Param tre indicant l'importance du temps total.
        epsilon : float
            Pr cision pour le calcul (default: 1e-5).
        N : int
            Nombre d' tapes  pour le calcul (default: 200).
        '''
        t_tilde = [t for t in np.linspace(0, x, N) if abs(self.position(t)-epsilon)>=
                                                      x][0]
        return self.evalue_cout_partiel(t_tilde)+lam*t_tilde

    def ameliore(self, N : int, epsilon : float, x : float, lam : float,
                                               configuration : Configuration):
        '''
        Algorithme qui am liore localement une fonction.

        Parameters
        ----------
        N : int
            Nombre d'it rations    calculer.
        epsilon : float
            Degr  de changement.
        x : float
            Position jusqu'  laquelle on veut am liorer la strat gie.
        lam : float
            Param tre utilis  pour calculer le co t total.
        configuration : Configuration
            Configuration pour laquelle on veut optimiser la strat gie.

        Returns
        -------
        Strategie
            Une nouvelle strat gie am lior e.

        Raises
        ------
        ValueError
            Si la strat gie initiale n'est pas valide.
        '''
        import copy
        if not(self.est_valide(configuration,x)):
            raise ValueError("La strat gie initiale n'est pas valide.")

        meilleure_strat = Strategie(self.vitesse)
        strategies = [meilleure_strat]

        for i in range(1, N):
            # False but works
            u = lambda t: self.vitesse(t)+epsilon*t
            nouvelle_strat = Strategie(u)
            if nouvelle_strat.evalue_cout_total(x, lam) < meilleure_strat.
                                                    evalue_cout_total(x, lam) and
                                                    nouvelle_strat.est_valide(
                                                    configuration, x):
                meilleure_strat = Strategie(u)
```

```
            epsilon = -epsilon/2
            strategies.append(meilleure_strat)

        for i in range(len(strategies)-1):
            if strategies[i+1].evalue_cout_total(x, lam)>strategies[i].
                                                        evalue_cout_total(x, lam):
                print("OHNONONONONONONONONONONONONO")
            if not(strategies[i].est_valide(configuration, x)):
                print("WHAT")
        return strategies


def plot_situation(ax : plt.axes, conf: Configuration, *args : list, tmax=0, xmax=0,
                                            epsilon=1e-5, N=10000, labels=[]):
    '''
    Produit une graphe de la situation en question.

    Notes
    -----
    On doit sp cifier soit la position ou le temps maximal.

    Parameters
    ----------
    ax : Axes
        Axes to draw to.
    conf : Configuration
        Configuration de feux    r presenter.
    *args : list
        Strat gies    r presenter.
    t_max : float
        Temps maximal    r presenter (default: 0).
    x_max : float
        Position maximale    r presenter (default: 0).
    epsilon : float
        Pr cision (default: 1e-5).
    N : int
        Nombre de divisions (default: 10000).
    labels : list
        Liste des noms pour les graphes.

    Returns
    -------
    plot
        Objet plot r presentant la situation.

    Raises
    ------
    ValueError
        Si le temps et la position maximale ne sont pas valides ou compatibles.
        Si les labels et les strat gies n'ont pas la m me longueur.
    '''
    if tmax == 0 and xmax == 0:
        raise ValueError("Sp cifiez un temps ou une position maximale.")
    if tmax<0 or xmax<0:
        raise ValueError("Le temps et la position maximale doivent  tre  positifs.")

    if len(labels) != 0 and len(labels) != len(args):
        raise ValueError("Les noms et les strat gies ne s'alignent pas.")

    if len(labels) == 0:
```

```
        labels = [""]*len(args)

    if xmax != 0:
        liste_temps = []
        for arg in args:
            t = 0
            while arg.position(t) < xmax:
                t += epsilon
            liste_temps.append(t)
        tmax = max(liste_temps)

    if tmax != 0:
        # Deg
        xmax = 10

    t = np.linspace(0, tmax, N)
    for n, arg in enumerate(args):
        if(not(isinstance(arg, Strategie))):
            raise TypeError("Cette fonction prend en argument des strat gies.")
        if not(arg.est_valide(conf, xmax)):
            ax.plot(t, list(map(arg.position, t)), linestyle="--", label=labels[n])
        else:
            ax.plot(t, list(map(arg.position, t)), label=labels[n])

    if labels != [""]*len(args):
        ax.legend()

    for feu in conf.feux:
        y = list(map(feu.fonction_graphe, t))
        ax.plot(t, y, c="red")

    ax.set_ylim(0)

    return ax


def positions_to_strategie(t_max : float, positions : np.array) -> Strategie:
    '''
    Calcule la strat gie qui se corr spond aux positions donn es.

    Parameters
    ----------
    t_max : float
        Temps d'arriv e.
    positions : np.array
        Positions de la voiture.

    Returns
    -------
    Strategie
        Meilleure correspondance aux positions donn es.

    Raises
    ------
    ValueError
        Si la strat gie r sultante n'est pas valide.
    '''
    temps = np.linspace(0, t_max, len(positions))

    def fonction_vitesse(t : float) -> float:
```

```
        i = 1
        while t<temps:
            i=i+1
        return (t[i]-t[i-1])/len(positions)

    return Strategie(fonction_vitesse)
```