Convex MPC trajectory planning – Verification issues

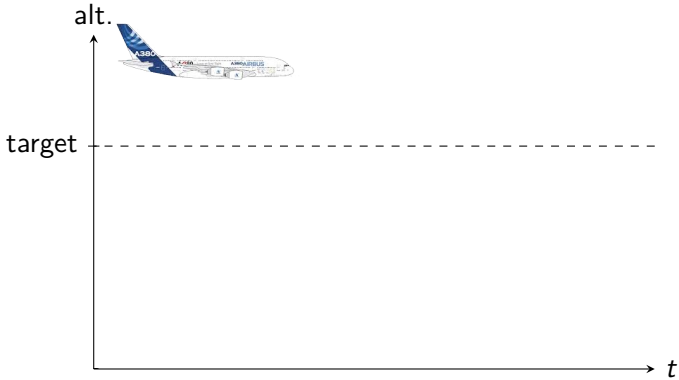
PL. Garoche

2019/02/05 – projet MRIS @ Polytechnique

Acnowledgment

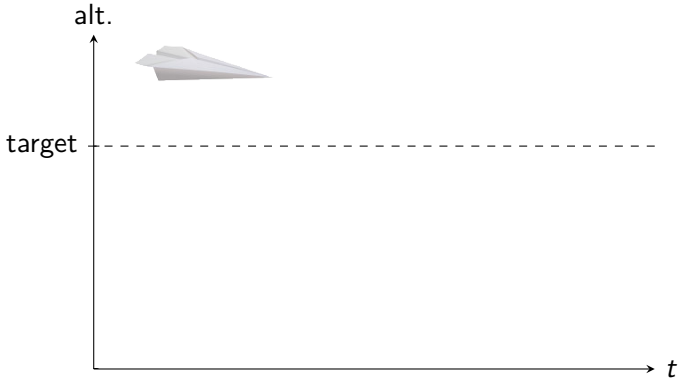
grand merci à Behçet Açikmese, Elliot Brendel, Alex Chapoutot, Guillaume Davy, Eric Féron, Didier Henrion, Bruno Hérissé, Pierre Vuillemin pour leur patience infinie à m'expliquer l'algèbre linéaire

How to control an aircraft trajectory?}



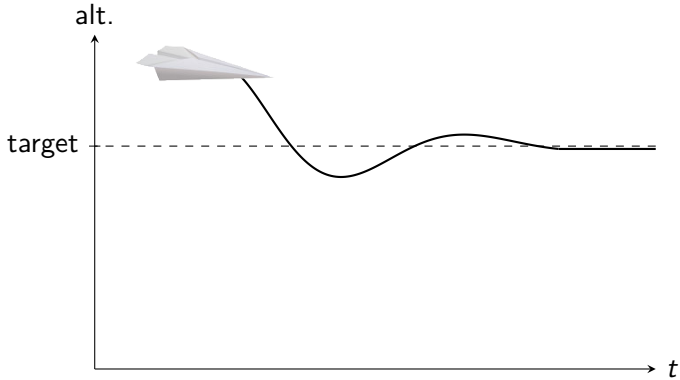
- feedback-loop: computing the "error" in position, eg. altitude target -1000 feet

How to control an aircraft trajectory?}



- feedback-loop: computing the "error" in position, eg. altitude target -1000 feet
- stable setting: simplifying the plant model

How to control an aircraft trajectory?}



- feedback-loop: computing the "error" in position, eg. altitude target -1000 feet
- stable setting: simplifying the plant model
- linear controller synthesis: eg. P/PD/PID/LQR/LPV/...

Basic solution: linear time-invariant controllers

Embedded Code:

- discretized version to be executed on a computer: digital fly-by-wire
- fairly simple: $x_{k+1} = Ax_k + Bu_k$
- in practice, a bit more complex: saturations, mode changes, etc. to improve performances and guarantee safety bounds.

Is it safe?

- the code evaluate in finite and predictable time
- strong mathematical evidences in the linear setting (frequency-domain analyses)
- used in commercial airline since 1984 (A320)!

How to improve performance/safety or control an unstable plant

Linear control are too limited to perform:

- aggressive maneuvering

[Acikmese, B. et al]

- propulsive planetary landing

[Blackmore, L.: Autonomous precision landing of space rockets.]

Optimal control vs receding horizon control

- Optimal control:
  - Rely on (possibly non linear) continuous dynamics.
  - Express the search for the trajectory as a bounded value problem
    - Choose input conditions determining control
    - Perform simulation and evaluate optimality conditions
    - Solution characterizes the continuous input function achieving the optimal trajectory
  - Pro/Cons:
    - Pro: Optimal solution
    - Cons: offline computation, trajectory (init/target points) have to be know a priori
- Receding Horizon / Model predictive control
  - Rely on the discrete time dynamics
  - Like Bounded Model checking + cost to optimize
  - Finite sequence of n steps, initial one is know, search for n inputs

Optimal control vs receding horizon control

- Optimal control:
  - Rely on (possibly non linear) continuous dynamics.
  - Express the search for the trajectory as a bounded value problem
    - Choose input conditions determining control
    - Perform simulation and evaluate optimality conditions
    - Solution characterizes the continuous input function achieving the optimal trajectory
  - Pro/Cons:
    - Pro: Optimal solution
    - Cons: offline computation, trajectory (init/target points) have to be know a priori
- Receding Horizon / Model predictive control
  - Rely on the discrete time dynamics
  - Like Bounded Model checking + cost to optimize
  - Finite sequence of n steps, initial one is know, search for n inputs

Rappel du contexte: calcul de trajectoire à la MPC

Encodage discret et linéaire de la dynamique du système:

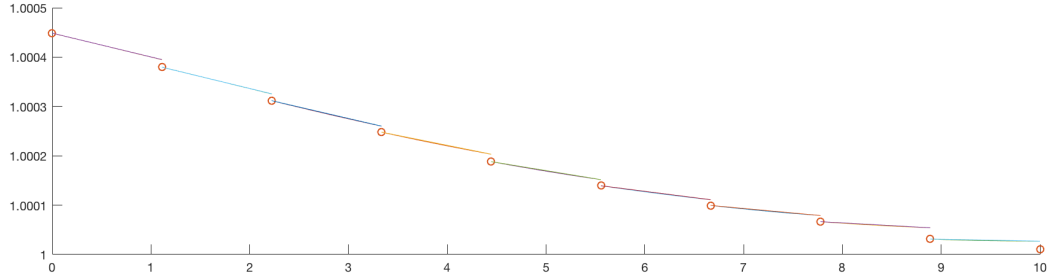
- N points
- entrée u_k constante sur l'intervalle de discrétisation
- $x_{k+1} = A_k x_k + B_k u_k + C_k$, $\forall k \in [0, N-1]$

Synthèse d'un problème d'optimisation convexe:

- plus fiable et meilleur convergence que le cas général
- contraintes assez expressives: ex. QP/SOCP, contraintes de la forme $||Ax + B||_2 \leq Cx + D$
- par contre, les seules contraintes linéaires convexes et concaves sont les relations linéaires: toute contrainte d'égalité est nécessairement linéaire y = A x + B u + C
- calcul en ligne de la solution!

Besoin de garanties de correction, précision et de performance, par ex. borner le nombre d'itérations ou d'opérations en virgule flottante.

Schéma général de l'algorithme




```
input: trajectoire initiale, ie. vecteur
       de points de fonctionnement   (x̂, û)
output: trajectoire calculee   (x̂, û)

tant que (pas converge)
 (At, Bt, Ct) = linearisation(xhat,uhat)
 (Ad, Bd, Cd) = discretisation(At, Bt, Ct, dt)
 xhat', uhat' = optimisation_convexe(Ad,Bd,Cd,...)
```

Implémentation: Matlab + CVX (librarie d'optimisation convexe)

Quid de la validité de l'implémentation de l'algorithme?

Partie continue: Modèle de Goddard – linéarisation

Dynamique non linéaire: $\dot{x} = \begin{pmatrix} \dot{r} & \dot{v} & \dot{m} \end{pmatrix}^\intercal = f \begin{pmatrix} r & v & m \end{pmatrix}^\intercal$

$$f \begin{pmatrix} r \\ v \\ m \end{pmatrix} = \begin{cases} v \\ -\frac{D(r,v)}{m} * \frac{v}{||v||} - grav(r) + C * \frac{u}{m} \\ -b * ||u||; \end{cases}$$

avec $grav(v) = \frac{1}{||r||^2} \frac{r}{||r||}$ et $D(r,v) = KD * ||v||^2 * e^{-kr*(||r||-1)}$

Calcul (manuel) des dérivées partielles:

$$\frac{\partial f}{\partial x}_{|\hat{x},\hat{u}} = \begin{pmatrix} 0 & Id(3) & 0 \\ \frac{\partial f}{\partial r}_{|\hat{x},\hat{u}} & \frac{\partial f}{\partial v}_{|\hat{x},\hat{u}} & \frac{\partial f}{\partial m}_{|\hat{x},\hat{u}} \\ 0 & 0 & 0 \end{pmatrix} \qquad \frac{\partial f}{\partial u}_{|\hat{x},\hat{u}} = \begin{pmatrix} 0 \\ C\frac{Id(3)}{\hat{m}} \\ -b\frac{\hat{u}^\intercal}{||\hat{u}||} \end{pmatrix}$$

$$\frac{\partial f}{\partial r}_{|\hat{x},\hat{u}} = \frac{KD\ kr}{\hat{m}}\ \hat{v}\ \hat{r}^\intercal\ \frac{||\hat{v}||}{||\hat{r}||}\ e^{-kr*(||\hat{r}||-1)} - (\frac{Id(3)}{||\hat{r}||^3} - \frac{3\ \hat{r}\ \hat{r}^\intercal}{||\hat{r}||^5})$$

$$\frac{\partial f}{\partial v}_{|\hat{x},\hat{u}} = -\frac{KD}{\hat{m}}(Id(3)||\hat{v}|| + \frac{3\ \hat{v}\ \hat{v}^\intercal}{||\hat{v}||})e^{-kr(||\hat{r}||-1)}$$

$$\frac{\partial f}{\partial m}_{|\hat{x},\hat{u}} = \frac{KD}{\hat{m}^2}\hat{v}\ ||\hat{v}||\ e^{-kr(||\hat{r}||-1)} - \frac{C}{\hat{m}^2}\ \hat{u})$$

$$\dot{x} \approx \underbrace{\frac{\partial f}{\partial x}_{|\hat{x},\hat{u}}}_{\text{At}} x + \underbrace{\frac{\partial f}{\partial u}_{|\hat{x},\hat{u}}}_{\text{Bt}} u + \underbrace{f(x) - \frac{\partial f}{\partial x}_{|\hat{x},\hat{u}} \hat{x} - \frac{\partial f}{\partial u}_{|\hat{x},\hat{u}} \hat{u}}_{\text{Ct}}$$

Discrétisation 1/2

Au point $(\hat{x}, \hat{u})$, on a At, Bt, Ct. On discrétise sur l'horizon [0, dt]:
Matrice état-transition d'un système linéaire: $\Phi(t', t)$: en partant de $x(t)$, on obtient $x(t') = \Phi(t', t)x(t)$

$$\begin{cases} \Phi(t,t) = Id \\ \frac{d\partial\Phi(\tau,t)}{\partial\tau} = At\Phi(\tau,t) \end{cases} \qquad \begin{cases} Ad = \Phi(t+dt,t) \\ Bd = Ad\int_t^{t+dt}\Phi(t,\tau)Bd\tau \\ Cd = Ad\int_t^{t+dt}\Phi(t,\tau)Cd\tau \end{cases}$$

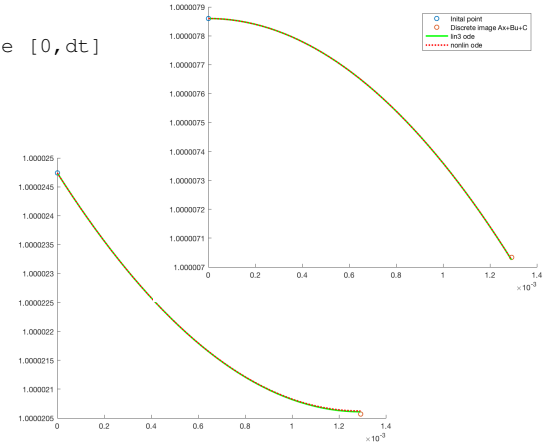
On a ici $\Phi(t,\tau) = \Phi^{-1}(\tau,t)$. Plus simplement, $\begin{cases} \Phi(t,t) = Id \\ \frac{d\partial\Phi(t,\tau)}{\partial\tau} = -\Phi(t,\tau,t)At \end{cases}$

- Algorithmes simples: Forward Euler

Discrétisation 2/2

Tous les calculs dans une seule boucle (en Matlab pour l'instant)

```
%% input: At. Bt. Ct. dt
%% output: Ad, Bd, Ct
resolution=100;
delta=dt/resolution;
% Calcul A, e^{-At}, B et C sur l'intervalle [0,dt]
Ad = eye(7);
Bd=zeros(7,3);
Cd=zeros(7,1);
Ard_k = eye(7);
for i=1:resolution
  Ad = Ad + delta * At * Ad;
  Ard = Ard - delta * Ard * At;
  Bd = Bd + (Ard * delta * Bt);
  Cd = Cd + (Ard * delta * Ct);
end
Bd = Ad * Bd;
Cd = Ad * Cd;
```



Optimisation convexe

Résolution du problème suivant:

variables: r(3,N), v(3.N), m(1.N), u(7,N) + variables utiles pour l'encodage
minimiser distance objectif - masse final
tel que
  % on suit la dynamique linearise
  $\forall$ k $\in$ [1,N-1], $(r,v,m)_{k+1} = Ad_k\ (r,v,m)_k + Bd_k\ u_k + Cd_k$
  % on satisfait des contraintes sur l'etat/l'entree
  $\forall$ k $\in$ [1,N], $\|u_k\| \leq 1$
  ...

- la durée de la trajectoire est fixée! ici N
- génération d'un code spécifique, typiquement point intérieur en primal/dual
  - difficile de garantir la convergence ou de prédire le nombre d'itération de l'algorithme
  - en pratique très rapide: < 10 itérations

Expérimentation: simulation en matlab

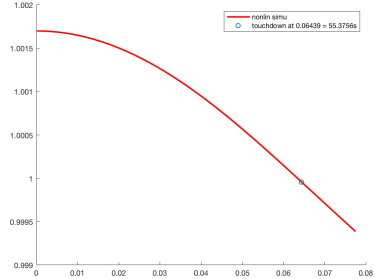
Problématique: avoir, dans un premier temps, une implémentation qui fonctionne sur un cas très simple

- simplification à l'extrême du problème: 1D, la fusée tombe tout droit
- identification des paramètres pertinents pour avoir des résultats satisfaisants
  - de nombreuses possibilités d'encoder les problèmes
- pas de solutions garanties mais recherche de paramètres pertinents, "sans erreurs numériques"

Scénario:

Quel nombre de pas de discrétisation?
Utilisation du cas simple: atterrissage dans 10s

- calcul: chute libre à partir de xinit
  puis touchdown -10 secondes
- utilisation de cet altitude pour
  construire une trajectoire avec
  soft-touchdown

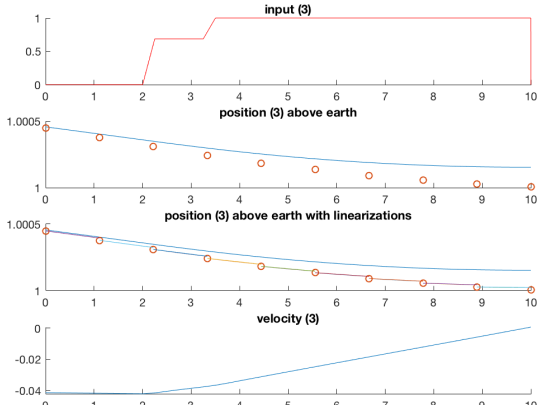


Paramètres de la méthode

- condition générale d'arrêt: un compteur + un notion de convergence sur $(\hat{x}, \hat{u})$
- finesse de l'intégration de la partie discrétisation
- changer l'algorithme: Backward Euler, Runge Kutta, utiliser A(t+dt)
- contraintes convexes
  - beaucoup de liberté dans l'encodage des contraintes
  - quel objectif à minimiser? faire évoluer l'echelle de temps (gain sur dt pour compresser/étendre l'échelle de temps)
  - expression des contraintes: utilisation de normes 2, par exemple écart entre x(final) et target, ou alors encadrer la poussée $||u|| \leq 1$ $\implies$ problème QP
  - pour les besoins de nos outils, évaluation encodage LP (moins naturel) par exemple $u_{1,2,3} \leq 1$ et $\sum_i u_i <= 1$

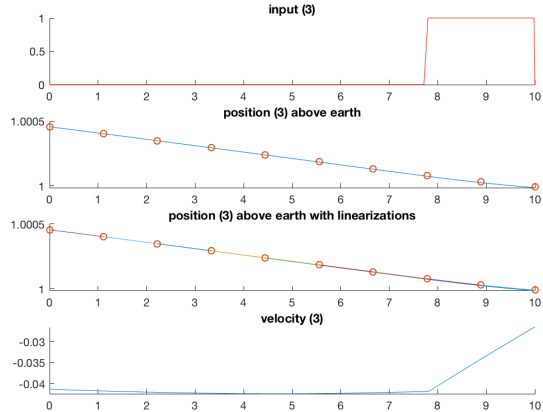
Encodage QP

```
minimize norm(rfinal - x(1:3,N),2)

x(:,1) == xinit;
for k=1:N-1
  x(:,k+1) == Ak{k} * x(:,k) + Bk{k} * u(:,k) + Ck{k} ;
end

for k=1:N-1
  norm(u(:,k)+u0) <= 1.0;
  x(3,k) >= 1.0
  x(7,k) >= 0.
  u(:,k) - u0 >= 0.0
end

% stop smoothly
norm(x(4:6,N)) <= 10e-10

% arrive at target
norm(x(1:3,N) - rfinal)<=10e-9
```



Encodage LP

```
minimize ones(1,3)*(rfinal-x(1:3,N))

x(:,1) == xinit;
for k=1:N-1
  x(:,k+1) == Ak{k} * x(:,k) + Bk{k} * u(:,k) + Ck{k} ;
end

for k=1:N-1
  u(1,k)+u(2,k)+u(3,k) <= 1.0;
  u(1,k)+u(2,k)+u(3,k) >= u0;
  u(1,k) >= 0;  u(2,k) >= 0;  u(3,k) >= 0;

  x(3,k) >= 1.0
  x(7,k) >= 0.
end

% stop smoothly
x(4,N) + x(5,N) + x(6,N) <= 10e-10
x(4,N)>=0;   x(5,N)>=0;  x(6,N)>= 0;

% arrive at target
ones(1,3)*(x(1:3,N) - rfinal)<=10e-8
```



Vers un encodage plus sophistiqué: éviter les problèmes

Un comportement plus précis:

- paramètre de compression/extension du temps
- variable de faisabilité
- poussée paramétrée en les poiints (pour l'instant constante sur l'intervalle)
- borner la différence entre $\hat{x}_{k-1}$ et $\hat{x}_k$ pour maitriser l'évolution des points
- augmenter la résolution de la discrétisation en temps (pour l'instant N=10)

Problèmes à l'exécution:

- parfois des valuations du problème calculées à la discrétisation posent des problèmes lors de l'optimisation: matrices singulières
- le calcul des dérivées partielles non défini pour certaines valeurs: eg. poussée nulle (division par ||u||)
  - on peut introduire un traitement spécifique:
    if u = 0 then xxx else xxx

Quid de la vérification?

Contributions sur la vérification d'algorithme d'optimisation convexe:

- méthode des points intérieurs, en primal pur, pour des problèmes LP, adaptable aux cônes plus sophistiqués;
- méthode des ellipsoïdes: moins performante en théorie, disponible en LP/QP/SOCP. Gestion des erreurs numériques.
- faisabilité de l'approche pour les deux méthodes.

Dans les deux cas, pour l'instant:

- prend un problème convexe: coût, contraintes A et b (dans cas linéaire)
- génération d'un code spécifique
- preuve que l'implémentation calcule une solution correcte, ϵ-optimale et un temps fini donné

Points intérieurs

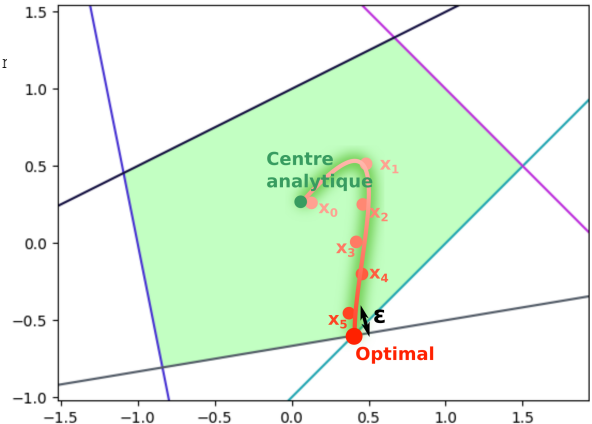
Schéma général de l'algorithme (pour LP)

- linérisation locale, séquence de pas de Newton
- pas: dépend du coût dans une norme basée Hessienne (dépend de la géométrie de l'ensemble faisable)

```
t=0; x=AC (centre analytique)
for i=1 to N
  G, gradient, H, hessienne, de la fonctior
```

$$t = t + \frac{\gamma}{\sqrt{c^T H^{-1} c}}$$
$$x = x - H^{-1}(G + ct)$$

Attention: appel à un algorithme de résolution de système linéaire



Point intérieurs LP primal garantis - application

Notre configuration:

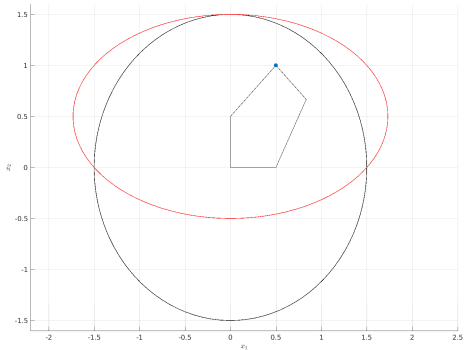
- problème fixé: A, b, c
  - intérieur non vide
  - espace faisable borné
- centre analytique fourni
- Choleski pour le calcul de H^{-1} (résolution des systèmes linéaires)

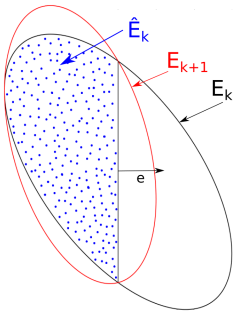
Automatiquement:

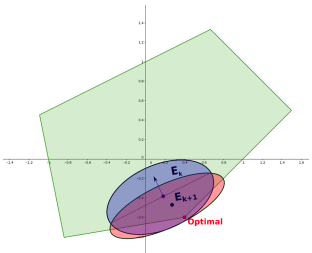
- calcul du nombre d'itérés requis pour l'optimalité
- génération du code et de sa preuve

Méthode des ellipsoïdes 1/2

$$Ell(B, c) = \{Bu + c : u^\intercal u \leq 1\}$$







Méthode des ellipsoïdes – algorithme et preuve de convergence 2/2

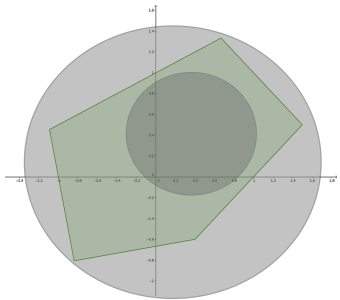
$$c_{k+1} = c_k - \frac{1}{(n+1)} \cdot B_k p \quad , \tag{1}$$

$$B_{k+1} = \frac{n}{\sqrt{n^2-1}} B_k + \left(\frac{n}{n+1} - \frac{n}{\sqrt{n^2-1}} \right) (B_k p) p^T \tag{2}$$

$$p = \frac{B_k^T e}{\sqrt{e^T B_k B_k^T e}}. \tag{3}$$

Prérequis sur l'ensemble faisable X et le coût f_o

- Une boule circonscrite R de centre x_C telle que $X \subset B_R(x_C)$
- Une boule inscrit r de centre x_c telle que $B_r(x_c) \subset X$
- Une borne sur le coût: un scalaire V tel que $\max_{x \in X} \; f_o(x) - \min_{x \in X} \; f_o(x) \leq V$



$\Rightarrow$ calcul du nombre d'itérés: $N = \text{ceil}\left(2n \cdot (n+1) \cdot \log(\frac{RV}{r\epsilon})\right)$

Pour aller plus loin: généraliser aux familles de problèmes

- Pour l'instant: génération et preuve pour une instance (A,b,c) donnée
- Extension à une famille d'instance
  - pour les ellipsoides, si on connait la variabilité des ens. faisable on peut calculer les ellipdoides générales pour toute cette famille.
  - pour le PI: pas de solution satisfaisante: uniquement modification du vecteur de contrainte ou du coût
    -> à creuser
- Comment calcule-t-on/borne-t-on cette variabilité? Analyse statique !

Matrices intervalles pour la linéarisation/discrétisation du calcul de trajectoire par méthode directe

Pour un point $(\hat{x}, \hat{u}) \in [x_{\min}, x_{\max}] \times [0,1]^3$

- $x_{\min}$: altitude ≥ 1, dry-mass;
- $x_{\max}$: altitude du point initial, masse initiale
- borner $f(\hat{x}, \hat{u})$ ainsi que At, Bt et Ct: $At^{\#}$, $Bt^{\#}$ et $Ct^{\#}$
  - peut-on spécialiser en fonction de k? si k=N alors $\hat{x} \approx x_{final}$

Calculer l'image, au sens matrice intervalle, pour un range de $(\hat{x}, \hat{u})$ donné

- arithmétique d'intervalles
- possibilité d'identifier les erreurs numériques accumulées
- ici les boucles ont un nombre d'itérations fini et connu à priori.

Questions:

- Impact de la structure des algorithmes sur la précision obtenue: à évaluer
- Comment calcule-t-on l'ellipsoide inscrite? le centre analytique?

Travaux à venir – Amélioration de l'encodage

Matlab – proposer un encodage performant

- étudier la paramétrisations des contraintes convexes
  - simulation des configurations et étude des solutions
  - eg. impact du pas de discrétisation sur
    - la solution calculée
    - la faisabilité de la preuve = la taille du système à optimiser
    - points nécessairement également répartis sur la trajectoire?
- scénario moins trivial: partir de plus haut, déplacement sur plusieurs dimensions
- export des instances de problèmes pour évaluer la génération de code prouvé

Application au projet MRIS

- appliquer l'algorithme aux drones du projet ?

Travaux à venir – Analyse statique – Vérification

Comment valider ces implémentations ?

Linéarisation / Discrétisation

- étudier les codes de linéarisation/discrétisation
- calculer les matrices intervalles
- cumul erreurs numérique ?

Travaux à venir – Analyse statique – Vérification

Comment valider ces implémentations ?

Linéarisation / Discrétisation

- étudier les codes de linéarisation/discrétisation
- calculer les matrices intervalles
- cumul erreurs numérique ?

Validation de l'optimisation convexe à priori

Comment valider les solveurs convexes pour un ensemble de problèmes d'entrée?

- étudier les solutions des problèmes intervalles et propager les invariants identifiés?
- impact sur correction, convergence ?
- garantir par construction que les problèmes analysés sont d'intérieur non vide

Travaux à venir – Analyse statique – Vérification

Comment valider ces implémentations ?

Linéarisation / Discrétisation

- étudier les codes de linéarisation/discrétisation
- calculer les matrices intervalles
- cumul erreurs numérique ?

Validation de l'optimisation convexe à priori

Comment valider les solveurs convexes pour un ensemble de problèmes d'entrée?

- étudier les solutions des problèmes intervalles et propager les invariants identifiés?
- impact sur correction, convergence ?
- garantir par construction que les problèmes analysés sont d'intérieur non vide

Vos idées sont les bienvenues!