Réglage des hyperparamètres pour la méthode Gauss-Seidel projetée dans les simulations de corps rigides

par

Marwen KRAIEM

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE DES TECHNOLOGIES DE L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 27 AVRIL 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Sheldon Andrews, Directeur de mémoire
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Adrien Gruson, Président du jury
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Rafael Menelau-Cruz, Membre du jury
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 13 AVRIL 2022

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Tout d'abord, je tiens à remercier mon directeur de maîtrise, Sheldon Andrews, qui a cru en moi, m'a encadré et m'a aidé à traverser les moments difficiles durant le cursus de ma maîtrise. Je tiens à le remercier chaleureusement pour les conseils qui m'ont aidé à me développer professionnellement et personnellement.

J'aimerais aussi remercier Henry Ho, Andreas Enzenhoefer et Paul Kry pour les commentaires et les conseils qui m'ont aidé à développer mes connaissances dans le domaine de la simulation physique. Je tiens également à remercier tous les membres du groupe de recherche de l'ÉTS pour leur soutien et leurs encouragements. Je tiens aussi à remercier les membres du groupe de recherche McGill-ÉTS-CM Labs Applied Dynamics pour leur intérêt et leurs commentaires à propos mon travail.

Je remercie tous les membres de ma famille. Leurs attentions et encouragements m'ont accompagnée tout au long de ces années. Enfin, je dédie ce mémoire à mes parents à qui je suis redevable pour leur soutien et leur sacrifice.

Les derniers mots de remerciement vont à mes amis et à tous mes collègues du Laboratoire de recherche multimédia.

Ce travail n'aurait pas été possible sans le soutien financer de l'organisme MITACS.

Réglage des hyperparamètres pour la méthode Gauss-Seidel projetée dans les simulations de corps rigides

Marwen KRAIEM

RÉSUMÉ

La simulation physique des corps rigides est affectée par le type de solveur utilisé pour résoudre les contraintes cinématiques. Pour les simulations interactives, il est recommandé d'utiliser un solveur interactif tel que la méthode de Gauss-Seidel projeté (PGS). Dans ce mémoire, un pipeline automatique est proposé pour optimiser certains paramètres de l'algorithme, notamment les paramètres de relaxation et de régularisation. L'optimisation des paramètres était réalisée en utilisant la méthode d'évolution d'adaptation de la matrice de covariance (CMA-ES).

Notre analyse confirme par une expérimentation rigoureuse que la directive proposée, pour le choix du paramètre de relaxation, dans des travaux précédents est un bon choix. On montre également que le paramètre SOR peut être ajusté sur une base par image par une relation linéaire avec le conditionnement des systèmes pour chaque simulation. Ces relations pourraient servir de base à des travaux futurs pour choisir le paramètre de relaxation à l'aide d'un ajustement basé sur l'apprentissage automatique pour une simulation donnée. De plus, le travail effectué nous a permis de réduire l'erreur globale grâce à l'utilisation d'hyperparamètres optimisés. La réduction de l'erreur s'accompagne également d'une réduction du nombre d'itérations. Ce dernier résultat pourrait être un point de départ pour réduire le temps de résolution dans les simulations interactives en temps réel.

Mots-clés: Simulation physique, corps rigide, PGS, optimisation, relaxation, régularisation, CMA-ES, analyse

Hyperparameter tuning for the projected Gauss-Seidel method in rigid body simulations

Marwen KRAIEM

ABSTRACT

Physical simulation of rigid bodies is affected by the type of solver used to solve the kinematic constraints. For interactive simulations, it is recommended to use an interactive solver such as the projected Gauss-Seidel (PGS) method. In this thesis, an automatic pipeline is proposed to optimize some parameters of the algorithm, including the relaxation parameter and the regularization parameter. The hyperparameters' optimization was performed using a covariance matrix adaptation evolution strategy (CMA-ES).

Our analysis confirms through rigorous experimentation that the guideline proposed, for the choice of the relaxation parameter, in previous works is a good choice. We also show that the SOR parameter can be adjusted on a per-frame basis by a linear relationship with the conditioning of the systems for each simulation. These relationships could serve as a basis for future work to choose the relaxation parameter using a machine learning-based adjustment for a given simulation. Moreover, the work done allowed us to reduce the overall error thanks to the use of optimized hyperparameters. The reduction of the error is also accompanied by a reduction of the number of iterations. This last result could be a starting point to reduce the resolution time in a real time interactive simulations.

Keywords: Physical simulation, rigid body, PGS, optimization, relaxation, regularization, CMA-ES, analysis

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ALGORITHMES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CMA-ES	Stratégies d'évolution avec adaptation de matrice de covariance / *Covariance matrix adaptation evolution strategy*
BLCP	Problème de complémentarité linéaire en boite / *Boxed linear complementarity problem*
LCP	Problème de complémentarité linéaire / *linear complementarity problem*
MLCP	Problème de complémentarité linéaire mixte / *mixed linear complementarity problem*
NLCP	Problème de complémentarité non-linéaire / *non-linear complementarity problem*
PGS	Gauss-Seidel projetée / *Projected Gauss-Seidel*
SIGGRAPH	Special Interest Group on Computer GRAPHics and Interactive Techniques (un séminaire international sur l'infographie et les techniques interactives)
SOR	Surrelaxation successive / *Successive over-relaxation*

LISTE DES SYMBOLES ET UNITÉS DE MESURE

$\mathbf{A}$	Matrice principale
$\mathbf{a}$	Vecteur d'accélération généralisée
α	Paramètre de régularisation
$\mathbf{b}$	Vecteur du côté droit de l'équation de mouvement sous contrainte
$\mathbf{C}$	Matrice de régularisation des contraintes
h	Pas du temps
e	Erreur de résolution
$\mathbf{f}$	Vecteur de force généralisée
$\mathbf{J}$	Matrice jacobienne
k_{max}	Nombre maximale d'itérations
κ	Paramètre de relaxation
$\mathbf{l}$	Vecteur de limite inférieure
$\boldsymbol{\lambda}$	Vecteur des impulsions de contrainte
$\mathbf{M}$	Matrice de masse
m	Nombre de contraintes
μ	Coefficient de frottement
n	Nombre des corps
$\mathbf{p}$	Vecteur de position
$\boldsymbol{\phi}$	Fonction des contraintes
$\mathbf{q}$	Vecteur des coordonnées généralisées
t	Variable temporelle (secondes)
$\boldsymbol{\theta}$	Quaternion d'orientation
$\mathbf{u}$	Vecteur de limite supérieure

$\mathbf{v}$	Vecteur de vitesse généralisé
$\mathbf{w}$	Vecteur des vitesses résiduelles
$\dot{\square}$	Dérivée de premier ordre par rapport au temps
$\ddot{\square}$	Dérivée de second ordre par rapport au temps
$\square^{+}$	Valeur implicite déterminée à la fin d'une étape d'intégration

INTRODUCTION

Les simulations physiques interactives sont souvent utilisées dans plusieurs domaines tels que la formation virtuelle, la production cinématographique et les jeux vidéo. Les animations incluses dans ce type d'applications contiennent dans la plupart du temps des simulations des multicorps rigides. Afin de disposer d'applications interactives et en temps réel, il est nécessaire de s'assurer que la fréquence d'images des simulations est adaptée à ce type d'application. Si la fréquence d'images des simulations ne peut pas être maintenue, cela peut créer un problème pour les applications où l'utilisateur est amené à interagir avec l'animation. Cela détruit l'expérience du réalisme dans des applications telles que les jeux et les formations en réalité virtuelle.

Dans ce cas, il est nécessaire de résoudre la dynamique de ces corps rigides en utilisant ce qu'on appelle les moteurs physiques de simulation. Avec ce type d'application, on cherche à produire des simulations stables, visuellement plausibles et interactives , en respectant des critères de performance (le nombre d'itérations, le temps de calcul, utilisation de mémoire, etc).

En outre, le compromis entre la stabilité, la plausibilité, la précision et l'interactivité est un grand défi pour avoir un bon moteur physique. C'est pourquoi qu'on trouve certaines applications qui favorisent un critère par rapport à l'autre. Par exemple, dans l'industrie des jeux vidéo, on cherche à garder une simulation interactive ou on peut sacrifier la précision dans plusieurs situations comme pour le cas de plusieurs jeux vidéo.

Dans les simulations physiques interactives, le choix de solveur est parmi les facteurs les plus importants affectant les performances de la simulation. En général, dans un tel type de simulation, on utilise souvent l'algorithme Projected Gauss Seidel (PGS) grâce à sa simplicité et sa rapidité de résolution de contraintes de mouvement. Aussi, c'est sur l'amélioration de cette méthode qu' on concentre nos efforts.

Motivation et objectives du mémoire

La résolution de la dynamique dans une simulation calculée dans les moteurs de jeu est un composant primordial pour avoir une simulation physique réelle et interactive. Les solveurs des contraintes des forces et de vitesses peuvent être un facteur déterminant quant à la performance de cette résolution.

En général, il existe deux types différents de solveurs numériques pour les équations de mouvement sous contraintes : les solveurs directs et les solveurs itératifs. Les solveurs directs sont bien adaptés pour traiter des systèmes linéaires numériquement difficiles et produire des solutions précises. Alors que les solveurs itératifs produisent rapidement un comportement plausible avec des solutions approximatives.

Dans ce mémoire, on se concentre sur les simulations interactives. Les solveurs itératifs qui sont plus populaires dans la création ce genre de simulations pour leur capacité à produire des solutions approximatives. La complexité de ce type de solveurs est beaucoup moins élevée que celle des solveurs directs. La complexité en temps des méthodes itératives tel que PGS est en général égale à $O(n^2)$, avec n le nombre de variables du système à résoudre. Mais une complexité proche de la complexité linéaire peut être atteinte pour des implémentations sans matrice. Par contre, les méthodes directes basées sur la factorisation de Cholesky ont une complexité de $O(n^3)$.

Il existe plusieurs techniques pour améliorer la convergence, par exemple, l'amélioration du conditionnement du système par l'ajout d'une régularisation ou de la relaxation. Parmi les solveurs itératifs les plus utilisés par les applications interactives, on trouve le solveur Gauss-Seidel projetée. Ce solveur se caractérise par une implémentation simple et une vitesse de résolution linéaire, ce qui permet d'avoir un comportement approximatif de la simulation

physique. Améliorer le taux de convergence de cet algorithme car cela permet d'avoir des approaches plus performantes.

Le travail effectué dans ce mémoire porte sur la recherche d'une analyse pertinente pour ajuster certains paramètres introduits dans l'algorithme PGS afin d'avoir un meilleur taux de convergence que celui existant. Dans la littérature, il est connu que le paramètre de relaxation favorise la convergence, ainsi que le paramètre de régularisation. Cependant, les valeurs de ces deux paramètres changent en fonction du système résolu à chaque pas de temps de la simulation et il est difficile d'ajuster ces valeurs en raison de la sensibilité de la convergence aux variations de ces deux paramètres.

Le premier objectif de notre recherche est le développement d'un pipeline automatique pour déterminer les valeurs optimales de paramètres introduits dans l'algorithme PGS. On cherche également à fournir une analyse pertinente qui produise un ensemble de règles pour la sélection de ces paramètres.

L'Organisation du mémoire

La suite de ce mémoire est organisée comme suit :

- **Chapitre 1 :** On résume la formulation théorique du problème dynamique de simulation des corps rigides. On présente aussi les méthodes numériques utilisées dans la résolution avec une concentration sur la méthode PGS.
- **Chapitre 2 :** On présente les modifications introduites dans l'algorithme de PGS ainsi que la méthodologie d'optimisation des paramètres introduit dans cet algorithme.
- **Chapitre 3 :** On présente les résultats de notre d'optimisation afin de déterminer les paramètres optimaux du PGS pour plusieurs simulations. On réalise une analyse des résultats et on compare les résultats de notre méthode à la méthode de référence.

- **Conclusion :** On produit nos déductions et conclusions finales ainsi que les travaux qui peuvent être réalisés dans le futur afin de bénéficier du travail réalisé.

CHAPITRE 1

ÉTAT DE L'ART

Dans ce chapitre, on présente le contexte théorique de la dynamique des corps rigides et les modèles utilisés ainsi que les différentes techniques employées pour résoudre les contraintes générées dans la simulation des corps rigides.

1.1 La formulation du problème

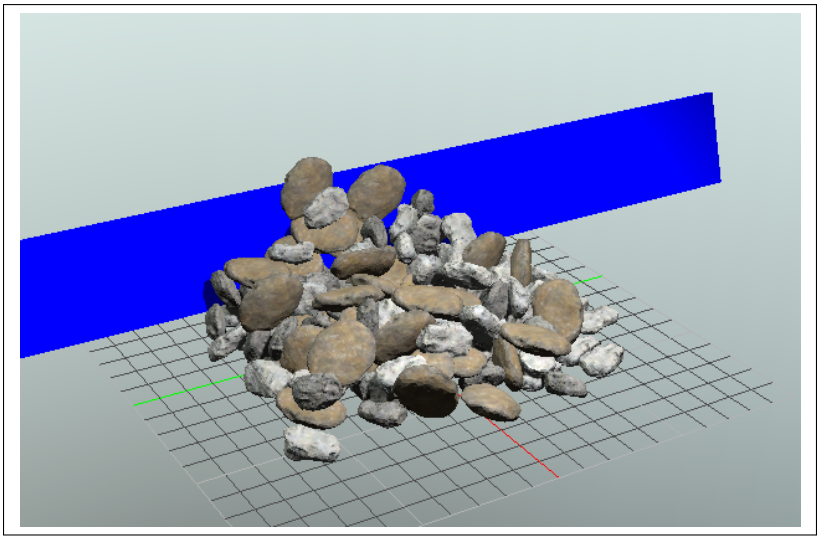


Figure 1.1 Exemple d'une simulation des corps rigides avec Vortex

Tout d'abord, un corps rigide est un solide dont la forme et la taille sont invariantes à toutes forces appliquées sur lui. Dans les simulations qui contiennent des corps rigides, comme dans la simulation présentée dans la figure 1.1, il est très probable qu'on va avoir des interactions. Il est important de mentionner que la figure 1.1 représente une capture de la simulation d'une lame de charrue qui pousse un tas de rochers. La masse et la géométrie des rochers sont modélisées dans le moteur physique de simulation *Vortex Dynamics*(CM Labs Simulations, 2019). Aussi, les contacts entre les rochers les font glisser, tomber et se pousser les uns contre les autres.

Ces interactions sont modélisées par des contraintes cinématiques diverses qui limitent le mouvement de ces corps les uns par rapport aux autres. Pour modéliser ces interactions, on doit résoudre les forces d'interface entre ces corps.

Dans la partie qui suit, on va décrire tous les éléments nécessaires pour formuler la simulation des corps rigides et plus particulièrement les interactions générées au cours de la simulation ainsi que les méthodes utilisées pour résoudre ces contraintes générées par ces interactions.

La formulation de la dynamique décrit dans les sections suivantes est la même que dans le moteur physique de simulation Vortex de CM Labs [1] qui est utilisée dans nos expérimentations.

Aussi, il convient de mentionner que les caractères italiques sont utilisés pour les valeurs scalaires, les sous-scripts pour les indices, les caractères gras pour les vecteurs et les matrices, les lettres majuscules pour les matrices et les lettres minuscules pour les vecteurs et les scalaires.

1.1.1 La cinématique des corps rigides

Dans l'espace réelle 3D, le mouvement d'un corps rigide décrit six degrés de libertés : trois pour préciser sa position dans l'espace et trois autres pour décrire son orientation.

La position du corps rigide $\mathbf{p} \in \mathbb{R}^3$ décrit les déplacements du corps par rapport à un repère de référence global définis par trois axes indépendants (voir figure 1.2).

Pour l'orientation d'un corps rigide, il y a une variété de modèles développés dans la littérature. (Erleben, 2005). Parmi les modèles existants, on trouve, les matrices de rotation et les quaternions de rotation. Vortex utilise des quaternions de rotation, $\boldsymbol{\theta} \in \mathbb{R}^4$, pour décrire l'orientation du corps rigide par rapport au repère de référence global. Un quaternion est formé de 4 scalaires, alors que seulement trois degrés de liberté sont nécessaires pour décrire le mouvement angulaire d'un corps dans un espace 3D. Par conséquent, un degré de liberté est supprimé par la contrainte selon laquelle tous les quaternions de rotation ont une longueur unitaire, de sorte que $\|\boldsymbol{\theta}\| = 1$.

[1] https://www.cm-labs.com/

L'utilisation des quaternions pour décrire les rotations des corps rigides est souvent choisie en raison de leurs propriétés bénéfiques. Par exemple, leurs représentations compactes offrent un grand avantage en termes de taille de stockage et de vitesse de calcul. De plus, elles résolvent le problème du blocage de cardan qui existe dans d'autres représentations telles que les angles d'Euler.

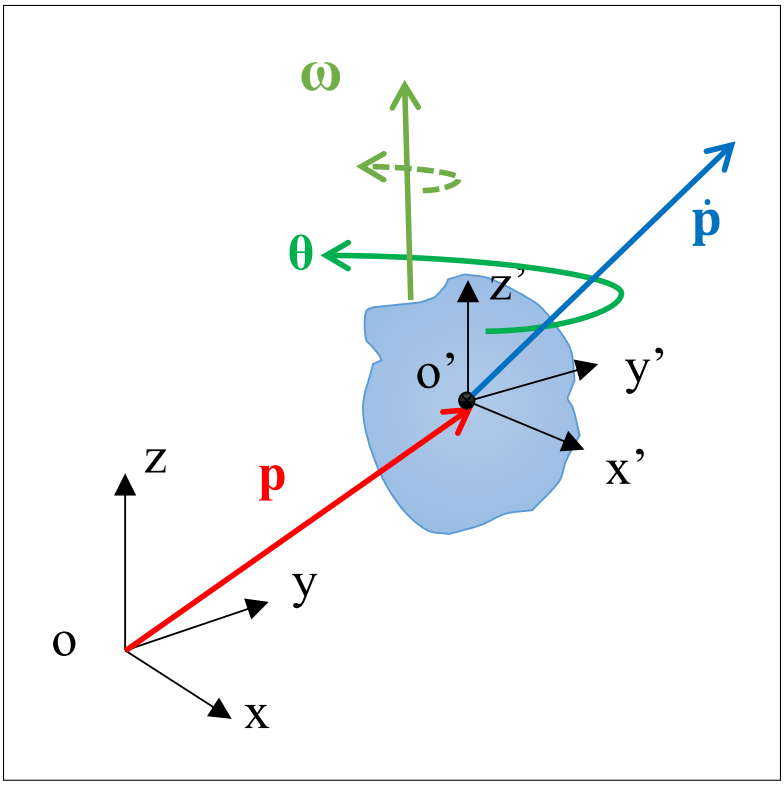


Figure 1.2 Illustration spatiale d'un corps rigide décrit par : une position $\mathbf{p}$, une orientation $\boldsymbol{\theta}$, une vitesse linéaire $\dot{\mathbf{p}}$ et une vitesse angulaire $\boldsymbol{\omega}$

Le groupement de la position et de l'orientation donne une description sur la configuration du corps rigide dans l'espace. On va noter la combinaison des quaternions et du vecteur de déplacement dans un seul vecteur $\mathbf{q} \in \mathbb{R}^7$ tel que :

$$\mathbf{q} = \begin{bmatrix} \mathbf{p} \\ \boldsymbol{\theta} \end{bmatrix} \tag{1.1}$$

Dans une simulation physique, la configuration spatiale du corps rigide change au cours du temps t. La dérivée de premier ordre par rapport au temps t de cette configuration $\mathbf{q}(t)$ donne une combinaison de deux vecteurs : $\dot{\mathbf{p}}(t) \in \mathbb{R}^3$ la vitesse linéaire et $\boldsymbol{\omega}(t) \in \mathbb{R}^3$ la vitesse angulaire. Le calcul de ce deux composantes se fait de la façon suivante :

$$\dot{\mathbf{p}}(t) = \frac{d\mathbf{p}(t)}{dt}$$

$$\boldsymbol{\omega}(t) = \mathbf{T}\frac{d\boldsymbol{\theta}(t)}{dt}$$

où $\mathbf{T} \in \mathbb{R}^{3\times4}$ est une matrice qui caractérise la cinématique du mouvement du corps rigide. La matrice $\mathbf{T}$ est nécessaire en raison de l'utilisation des quaternions pour représenter les rotations. La vitesse généralisée du corps s'écrit sous la forme suivante :

$$\mathbf{v}(t) = \begin{bmatrix} \dot{\mathbf{p}}(t) \\ \boldsymbol{\omega}(t) \end{bmatrix} \tag{1.2}$$

La dérivée de second ordre par rapport au temps t de la configuration du corps rigide donne l'accélération du corps :

$$\mathbf{a}(t) = \begin{bmatrix} \ddot{\mathbf{p}}(t) \\ \dot{\boldsymbol{\omega}}(t) \end{bmatrix} \tag{1.3}$$

Pour un ensemble de n corps rigide, on a trois vecteurs décrivant tout le système : $\mathbf{q}(t) \in \mathbb{R}^{7n}$, $\mathbf{v}(t) \in \mathbb{R}^{6n}$ et $\mathbf{a}(t) \in \mathbb{R}^{6n}$. On redéfinit ces trois vecteurs en concaténant les quantités pour chaque corps en un vecteur global pour chacune des positions, vitesses et accélérations du système, tel que

$$\begin{aligned} \mathbf{q}(t) &= \left[\mathbf{q}_1^\top(t) \dots \mathbf{q}_n^\top(t)\right]^\top \\ \mathbf{v}(t) &= \left[\mathbf{v}_1^\top(t) \dots \mathbf{v}_n^\top(t)\right]^\top \\ \mathbf{a}(t) &= \left[\mathbf{a}_1^\top(t) \dots \mathbf{a}_n^\top(t)\right]^\top \end{aligned} \tag{1.4}$$

1.1.2 Les équations du mouvement

L'équation de mouvement de Newton-Euler pour la simulation des corps rigides, comme décrit par Bender, Erleben & Trinkle (2014), permet de modéliser la dynamique du système physique des n corps rigides en mouvement. Cette équation s'écrit sous la forme suivante :

$$\mathbf{M}\mathbf{a}(t) = \mathbf{f}(t) \tag{1.5}$$

où $\mathbf{M} \in \mathbb{R}^{6n \times 6n}$ est la matrice de masse et d'inertie, $\mathbf{a}(t) \in \mathbb{R}^{6n}$ est le vecteur d'accélération combinant les accélérations de tous les corps rigides et $\mathbf{f}(t) \in \mathbb{R}^{6n}$ est le vecteur de somme des forces.

1.1.3 Intégration temporelle

La simulation physique est une séquence temporelle discrète des configurations du système. Pour passer d'un état a une autre, on doit incrémenter le temps d'*un pas de temps* h où $0 < h \leq 1$ s. On utilise une valeur très faible, proche de zéro, pour le pas de temps h. Cela nous permet d'utiliser une approximation à la série de Taylor du premier ordre pour décrire la vitesse pour la prochaine pas temporelle en fonction la vitesse actuelle et l'accélération actuelle $\mathbf{v}(t+h) \approx \mathbf{v}(t) + \mathbf{a}(t)h$. On va noter $\mathbf{v}^+ = \mathbf{v}(t+h)$ et $\mathbf{v} = \mathbf{v}(t)$. L'utilisation de ces notations, nous permet de se débarrasser de la notation (t), utilisée pour définir la fonction dépendante du temps. De plus, l'utilisation du symbole + en exposant sert à indiquer le statut implicite du vecteur.

L'accélération du système s'écrit comme suit :

$$\mathbf{a}(t) = \frac{\mathbf{v}^+ - \mathbf{v}}{h} \tag{1.6}$$

En remplaçant l'accélération dans l'équation (1.5) par l'expression (1.6), on obtient l'équation suivante :

$$\mathbf{M}(\mathbf{v}^+ - \mathbf{v}) = h\mathbf{f} \tag{1.7}$$

1.1.4 Les contraintes cinématiques

Au cours d'une simulation des corps rigides, on doit tenir compte des contraintes cinématiques afin de créer un mouvement cohérent entre les corps dans notre animation. Une contrainte cinématique est une fonction qui donne l'erreur générée lors la violation d'une contrainte sur un/plusieurs degré(s) de liberté. Par exemple, dans le cas d'une charnière reliant deux objets solides, les deux objets tournent l'un par rapport à l'autre autour d'un axe de rotation fixe. Tout autre mouvement, autre que la rotation autour de cet axe fixe, est considéré comme une violation des contraintes de la charnière.

Soit un ensemble de m fonctions de contraintes $\phi(\mathbf{q}, t) \in \mathbb{R}^m$ qui définissent le mouvement des corps dans un espace ayant $6n$ degrés de libertés.

Dans une simulation physique, pour décrire les contraintes de mouvement entre les corps rigides, on utilise deux types de contraintes cinématiques : *bilatérales* et *unilatérales*.

Les contraintes bilatérales (Exemples : charnière, articulation sphérique) ont la forme :

$$\phi(\mathbf{q}, t) = 0 \tag{1.8}$$

Les contraintes unilatérales s'écrivent sous la forme suivante :

$$\phi(\mathbf{q}, t) \geq 0 \tag{1.9}$$

Les contraintes écrites ci-dessus sont des contraintes exprimées en fonction de la position. On va les transformer, en calculant le gradient de $\phi(\mathbf{q}, t)$, en des contraintes de vitesse pour avoir une formulation homogène avec l'équation (1.7).

En appliquant le théorème de dérivation des fonctions composées sur la fonction $\phi(\mathbf{q}, t)$, on peut écrire la dérivée de premier ordre de la fonction de contrainte de la manière suivante :

$$\dot{\phi}(\mathbf{q}, t) = \frac{d\phi}{dt} = \frac{\partial \phi}{\partial \mathbf{q}} \frac{d\mathbf{q}}{dt} = \mathbf{J}\mathbf{v}$$

où $\mathbf{v} = \frac{d\mathbf{q}}{dt}$ la vitesse et $\mathbf{J} = \frac{\partial \phi}{\partial \mathbf{q}} = [\frac{\partial \phi_1}{\partial \mathbf{q}} \ \frac{\partial \phi_2}{\partial \mathbf{q}} \ \dots \ \frac{\partial \phi_m}{\partial \mathbf{q}}] \in \mathbb{R}^{m \times 6n}$ est la matrice jacobienne formée par les gradients des fonctions des contraintes du système.

Ainsi, l'équation des contraintes bilatérales de vitesse s'écrit sous la forme suivante :

$$\mathbf{Jv} = \mathbf{0} \tag{1.10}$$

et de même pour les contraintes unilatérales :

$$\mathbf{Jv} \geq \mathbf{0} \tag{1.11}$$

1.1.5 Les contraintes de contact

Les contacts entre deux ou plusieurs corps rigides sont des contraintes très récurrentes dans les simulations des corps rigides. Pour modéliser et comprendre en détail ce type de contraintes, on va définir les modèles utilisées pour les décrire ainsi que le modèle utilisé pour simuler le frottement entre ces corps.

1.1.5.1 Modélisation des contraintes de contact

Pour modéliser les contacts dans notre simulation de corps rigides, on utilise des contraintes unilatérales $\phi(\mathbf{q}, t) \geq 0$. Les contacts seront actifs seulement lorsqu'il y a collision entre les deux corps ($\phi(\mathbf{q}, t) \leq 0$). Dans le cas où ϕ est strictement positive, les deux corps seront donc séparés (figure 1.3).

On suppose que nous avons des corps lisses afin de définir deux vecteurs unitaires perpendiculaires aux surfaces de contact des deux corps. Ensuite, nous définissons notre plan de contact qui est décrit par le point de contact et un vecteur unitaire parallèle à l'un des deux vecteurs normaux des surfaces de deux corps en contact (voir figure 1.5).

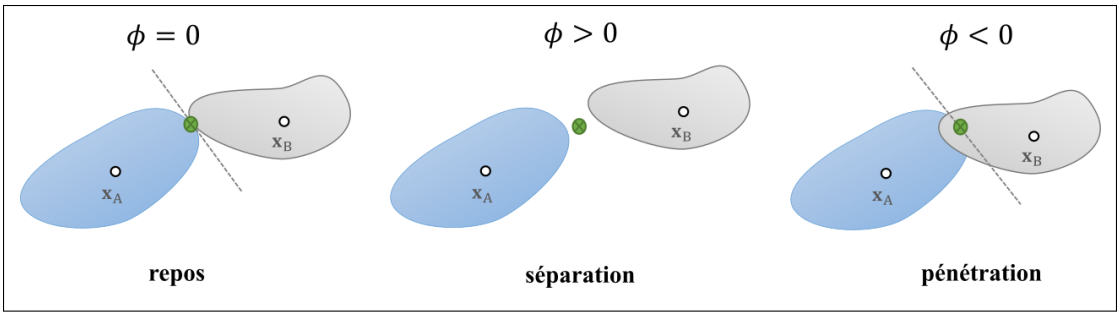


Figure 1.3 Illustration des trois états de contact classés selon la valeur de la fonction de contrainte ϕ

Comme point de départ, on ne va pas considérer le frottement pour le moment. On prend donc une force $\mathbf{f}_c = \lambda_{\vec{\mathbf{n}}}\vec{\mathbf{n}}$, perpendiculaire au plan de contact, qui sera appliquée au point de contact c afin d'empêcher les corps de s'interpénétrer. On note que la force $\mathbf{f}_c$ est décomposée en une magnitude, $\lambda_{\vec{\mathbf{n}}}$, et une direction $\vec{\mathbf{n}}$.

Pour appliquer une telle force sur les corps dans les simulations, on doit tenir compte de deux considérations possibles (Andrews & Erleben, 2021) :

- Soit les corps ne sont plus en contact :

$$\phi > 0, \ \lambda_{\vec{\mathbf{n}}} = 0$$

- Soit les corps sont en contact et on doit appliquer une force non nulle au point de contact :

$$\phi = 0, \ \lambda_{\vec{\mathbf{n}}} > 0$$

Ces deux conditions peuvent être écrites en une seule expression exprimant la complémentarité des conditions de contact :

$$0 \leq \phi \perp \lambda_{\vec{\mathbf{n}}} \geq 0 \tag{1.12}$$

L'équation (1.12) exprime la condition de complémentarité au niveau de position. Puisqu'on va utiliser une formulation de contraintes à base de vitesse, on va extraire la condition de complémentarité de non-pénétration à base de la vitesse.

Cette condition s'écrit à l'aide de l'équation 1.13.

$$0 \leq \dot{\phi} \perp \lambda_{\vec{\mathbf{n}}} \geq 0, \tag{1.13}$$

avec $\dot{\phi}$ est la vitesse relative entre les deux corps en contact au point $\mathbf{p}_c$.

Dans la suite, on va noter $\dot{\phi}$ comme $w_{\vec{\mathbf{n}}}$. La prochaine étape sera de définir l'expression analytique de la vitesse relative entre deux corps en contact.

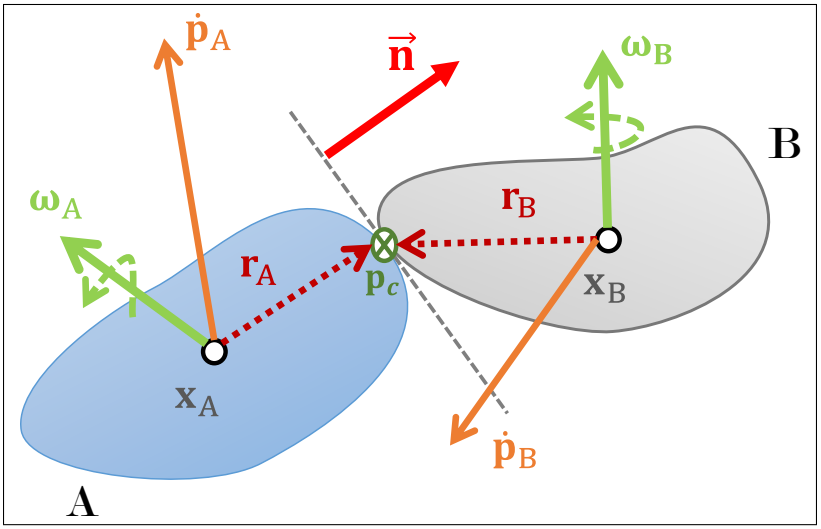


Figure 1.4 Notations impliquées dans le calcul de la vitesse relative du point de contact $\mathbf{p}_c$ entre deux corps rigides

$w_{\vec{\mathbf{n}}}$ sera la vitesse de changement de la contrainte cinématique $\phi(\mathbf{q}, t)$, on peut donc l'écrire de la manière suivante :

$$w_{\vec{\mathbf{n}}} = \frac{d\phi}{dt} = \frac{\partial \phi}{\partial \mathbf{q}} \frac{d\mathbf{q}}{dt} = \mathbf{J}\mathbf{v} \tag{1.14}$$

Maintenant, si on considère deux corps rigides A et B ayant respectivement les centres de masse $\mathbf{x}_A$ et $\mathbf{x}_B$ et qui entrent en collision au point $\mathbf{p}_c$ avec un vecteur normal pointant de A vers B (voir figure 1.4). Une autre expression de $w_{\vec{\mathbf{n}}}$ est la suivante en considérant que cette vitesse est la

différence entre les vitesses de A et de B au point $\mathbf{p}_c$:

$$w_{\vec{\mathbf{n}}} = \vec{\mathbf{n}}^\top \Delta\mathbf{v} \tag{1.15}$$

où $\Delta\mathbf{v}$ est la vitesse relative du point de contact $\mathbf{p}_c$:

$$\Delta\mathbf{v} = (\mathbf{v}_B + \omega_B \times (\mathbf{p}_c - \mathbf{x}_B)) - (\mathbf{v}_A + \omega_A \times (\mathbf{p}_c - \mathbf{x}_A)) \tag{1.16}$$

où $\mathbf{v}_A$ et $\mathbf{v}_B$ sont les vitesses linéaires des centres de masse et $\boldsymbol{\omega}_A$ et $\boldsymbol{\omega}_B$ sont les vitesses angulaires.

Enfin, on a une expression de $w_{\vec{\mathbf{n}}}$ formée par le produit de la matrice jacobienne $\mathbf{J}$ et le vecteur vitesse du système des corps en contact $\mathbf{v}$:

$$w_{\vec{\mathbf{n}}} = \underbrace{\begin{bmatrix} -\vec{\mathbf{n}}^\top & \vec{\mathbf{n}}^\top \mathbf{r}_{\mathrm{A}}^\times & \vec{\mathbf{n}}^\top & -\vec{\mathbf{n}}^\top \mathbf{r}_{\mathrm{B}}^\times \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \dot{\mathbf{p}}_{\mathrm{A}} \\ \boldsymbol{\omega}_{\mathrm{A}} \\ \dot{\mathbf{p}}_{\mathrm{B}} \\ \boldsymbol{\omega}_{\mathrm{B}} \end{bmatrix}}_{\mathbf{v}} \tag{1.17}$$

où $\mathbf{r}_{\mathrm{A}}^\times$ et $\mathbf{r}_{\mathrm{B}}^\times$ sont deux matrices antisymétriques associées respectivement aux vecteurs $\mathbf{r}_{\mathrm{A}} = (\mathbf{p}_c - \mathbf{x}_{\mathrm{A})}$ et $\mathbf{r}_{\mathrm{B}} = (\mathbf{p}_c - \mathbf{x}_{\mathrm{B}})$.

On introduit le frottement dans notre expression analytique de vitesse relative entre les deux corps. On va supposer qu'on a un repère de contact $[\vec{\mathbf{n}}\ \vec{\mathbf{t}}\ \vec{\mathbf{b}}]$ situé au point $\mathbf{p}_c$. Notant que le modèle de frottement sera décrit plus en détail dans la section suivante (1.1.5.2). La condition de complémentarité après l'intégration du frottement dans notre modèle s'écrit de la manière suivante en notant $\boldsymbol{\lambda} = [\lambda_{\vec{\mathbf{n}}}\ \lambda_{\vec{\mathbf{t}}}\ \lambda_{\vec{\mathbf{b}}}]^\top$:

$$0 \leq \mathbf{w} \perp \boldsymbol{\lambda} \geq 0, \tag{1.18}$$

où la vitesse relative $\mathbf{w}$ du contact au point $\mathbf{p}_c$ s'écrit sous la forme suivante (Andrews & Erleben, 2021) :

$$\mathbf{w} = \underbrace{\left[\vec{\mathbf{n}}\ \vec{\mathbf{t}}\ \vec{\mathbf{b}}\right]^\top}_{\mathbf{D}^\top} \Delta\mathbf{v} \tag{1.19}$$

Enfin, en s'inspirant de l'équation (1.17), nous avons une équation généralisée de la vitesse relative du point de contact entre deux corps A et B :

$$\mathbf{w} = \underbrace{\mathbf{D}^\top \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{r}_\mathrm{A}^\times & \mathbf{I}_{3\times3} & -\mathbf{r}_\mathrm{B}^\times \end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix} \dot{\mathbf{p}}_\mathrm{A} \\ \boldsymbol{\omega}_\mathrm{A} \\ \dot{\mathbf{p}}_\mathrm{B} \\ \boldsymbol{\omega}_\mathrm{B} \end{bmatrix}}_{\mathbf{v}} \tag{1.20}$$

1.1.5.2 Modélisation du frottement

Simuler les frottements dans les applications graphiques rend les simulations plus réelles. Pour introduire le frottement dans notre modèle de simulation physique, on doit introduire des contraintes de mouvement supplémentaires qui permettent de tenir compte de la force appliquée dans le plan de contact pour ralentir ou arrêter le mouvement d'un corps par rapport à l'autre. Cette force de frottement tangentiel s'écrit de la manière suivante :

$$\boldsymbol{\lambda}_{\vec{\mathbf{f}}} = \lambda_{\vec{\mathbf{t}}}\vec{\mathbf{t}} + \lambda_{\vec{\mathbf{b}}}\vec{\mathbf{b}} \tag{1.21}$$

où $\vec{\mathbf{t}}$ et $\vec{\mathbf{b}}$ sont deux vecteurs unitaires décrivant le plan de contact et qui forment avec $\vec{\mathbf{n}}$ une base orthonormée et $\lambda_{\vec{\mathbf{t}}}$, $\lambda_{\vec{\mathbf{b}}}$ sont deux scalaires. Le frottement sera représenté à l'échelle macroscopique par *la loi de Coulomb*. Dans cette loi, on distingue deux types de frottement possibles : *frottement statique* et *frottement dynamique*.

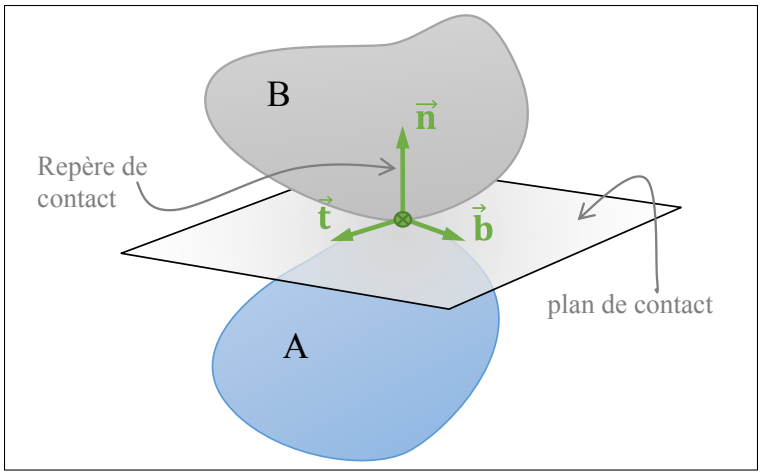


Figure 1.5 Plan et repère de contact

Dans la phase de frottement statique, on applique une force sur le corps pour maintenir son état statique. La force de frottement tangentiel $\boldsymbol{\lambda}_{\vec{\mathbf{f}}}$ est limitée par le produit du coefficient de frottement statique μ_s et de l'amplitude de la force normale $\boldsymbol{\lambda}_{\vec{\mathbf{n}}}$:

$$||\boldsymbol{\lambda}_{\vec{\mathbf{f}}}|| \leq \mu_s ||\boldsymbol{\lambda}_{\vec{\mathbf{n}}}|| \tag{1.22}$$

La force de contact total appliquée sera la somme de la force de frottement et la force normale au plan de contact. les forces de contact réalisables peuvent être représentées par un cône de frottement comme dans la figure 1.6.

Dans la phase dynamique, le corps glisse progressivement sur la surface et la force de frottement sert à ralentir le mouvement du corps par rapport l'autre. L'amplitude de la force de frottement $\boldsymbol{\lambda}_{\vec{\mathbf{f}}}$ sera égale à l'amplitude de la force normale $\boldsymbol{\lambda}_{\vec{\mathbf{n}}}$ multipliée par le coefficient de frottement dynamique μ_d :

$$||\boldsymbol{\lambda}_{\vec{\mathbf{f}}}|| = \mu_d ||\boldsymbol{\lambda}_{\vec{\mathbf{n}}}|| \tag{1.23}$$

Dans cette thèse, on va considérer que $\mu_s = \mu_d = \mu$.

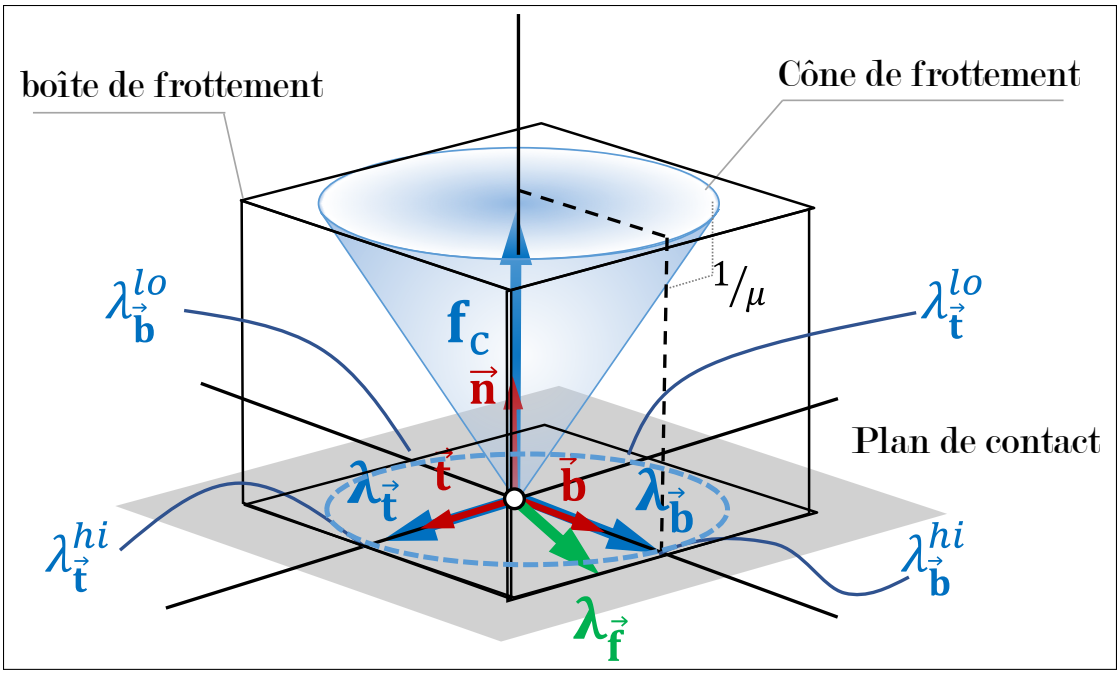


Figure 1.6 Visualisation des forces de contact réalisables dans le cône de frottement approximé par une boite de frottement

Afin d'avoir un modèle simple à résoudre, nous allons approximer le cône de friction par une boîte. Par conséquent, on obtient une formulation linéaire du frottement avec la discrétisation de la direction de la force de frottement en utilisant deux directions tangentes orthogonales $\vec{\mathbf{t}}$ et $\vec{\mathbf{b}}$. Cela permet d'écrire des inégalités indépendantes dans chaque direction de frottement pour avoir une force de glissement limitée par le coefficient de frottement, μ, multiplié par la force normale, $\lambda_{\vec{\mathbf{n}}}$.

$$\begin{aligned} \lambda_{\vec{\mathbf{t}}}^{lo} \leq \lambda_{\vec{\mathbf{t}}} \leq \lambda_{\vec{\mathbf{t}}}^{hi} \\ \lambda_{\vec{\mathbf{b}}}^{lo} \leq \lambda_{\vec{\mathbf{b}}} \leq \lambda_{\vec{\mathbf{b}}}^{hi} \end{aligned} \tag{1.24}$$

On définit $\boldsymbol{\lambda}^{lo} = [0 \underbrace{(-\mu\lambda_{\vec{\mathbf{n}}})}_{\lambda_{\vec{\mathbf{t}}}^{lo}} \underbrace{(-\mu\lambda_{\vec{\mathbf{n}}})}_{\lambda_{\vec{\mathbf{b}}}^{lo}}]^\top$ et $\boldsymbol{\lambda}^{hi} = [\infty \underbrace{(\mu\lambda_{\vec{\mathbf{n}}})}_{\lambda_{\vec{\mathbf{t}}}^{hi}} \underbrace{(\mu\lambda_{\vec{\mathbf{n}}})}_{\lambda_{\vec{\mathbf{b}}}^{hi}}]^\top$ deux vecteurs qui décrivent la boite de frottement.

La force de frottement $\boldsymbol{\lambda}_{\vec{\mathbf{f}}}$ est égale au vecteur $[\lambda_{\vec{\mathbf{t}}}\ \lambda_{\vec{\mathbf{b}}}]^\top$, où $\lambda_{\mathbf{t}}$ et $\lambda_{\vec{\mathbf{b}}}$ sont respectivement les projections de $\boldsymbol{\lambda}_{\vec{\mathbf{f}}}$ sur les vecteurs unitaires $\vec{\mathbf{t}}$ et $\vec{\mathbf{b}}$.

Ces deux vecteurs nous permettent aussi d'écrire la condition de faisabilité sur la force de contact recherchée (on va considérer que dans la plupart des cas $\boldsymbol{\lambda}^{lo} < \boldsymbol{\lambda}^{hi}$) :

$$\boldsymbol{\lambda}^{lo} \leq \boldsymbol{\lambda} \leq \boldsymbol{\lambda}^{hi} \tag{1.25}$$

D'après le modèle de friction de Coulomb, le frottement dynamique se réalise lorsque la vitesse relative au point de contact à une valeur non nulle. Dans ce cas, la force de frottement $\boldsymbol{\lambda}_f$ atteint sa valeur maximale et prend une direction opposée au mouvement.

Cela se traduit par les relations suivantes :

$$\begin{aligned} 0 \leq \mathbf{w}_l \perp \boldsymbol{\lambda} - \boldsymbol{\lambda}^{lo} \geq 0 \\ 0 \leq \mathbf{w}_u \perp \boldsymbol{\lambda}^{hi} - \boldsymbol{\lambda} \geq 0 \\ 0 \leq \mathbf{w}_u \perp \mathbf{w}_l \geq 0 \end{aligned} \tag{1.26}$$

avec $\mathbf{w} = \mathbf{w}_u - \mathbf{w}_l$ est une séparation de la vitesse au point de contact $\mathbf{p}_c$ comme la différence de deux quantités positives.

Les équations (1.26) sont les équations qui décrivent la complémentarité et la faisabilité des forces de contact au cours de la simulation.

1.1.6 Formulation dynamique

1.1.6.1 Équation de mouvement sous contraintes

Les contraintes cinématiques peuvent être imposées en appliquant des forces et des couples. Ces forces vont agir dans les directions décrites par la matrice jacobienne $\mathbf{J}$ (le gradient de $\phi(\mathbf{q}, t)$) afin de maintenir les contraintes de mouvement. Ainsi, on peut écrire les forces de contraintes sous la forme

$$\mathbf{f}_c = \mathbf{J}^\top \boldsymbol{\lambda}, \tag{1.27}$$

où $\boldsymbol{\lambda}$ est le multiplicateur de Lagrange interprété comme la valeur de cette force appliquée dans la direction $\mathbf{J}$.

Pour corriger les violations des contraintes de mouvement ($\phi(\mathbf{q}, t) = 0$ et $\phi(\mathbf{q}, t) \geq \mathbf{0}$) au cours de la simulation, on va introduire les impulsions $h\mathbf{f}_c$ dans l'équation (1.28). Afin d'éviter la répétition d'écriture de h dans les équations, on va considérer que $\boldsymbol{\lambda}$ est équivalente à $h\boldsymbol{\lambda}$. L'équation de mouvement (1.7) peut être écrite de la manière suivante :

$$\mathbf{M}\mathbf{v}^+ - \mathbf{J}^\top \boldsymbol{\lambda}^+ = \mathbf{M}\mathbf{v} + h\mathbf{f}. \tag{1.28}$$

Ici, $\boldsymbol{\lambda}^+ \in \mathbb{R}^m$ est le vecteur d'impulsion des contraintes à la prochaine pas du temps.

Il faut noter que les contraintes seront appliquées en utilisant les vitesses implicites, $\mathbf{v}^+$, et elles font partie du système linéaire utilisé pour résoudre les vitesses à la fin du pas de temps actuel.

Maintenant, en assemblant l'équation (1.28) et l'équation (1.10), on obtient une équation de mouvement sous contraintes :

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^\top \\ \mathbf{J} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{M}\mathbf{v} + h\mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{1.29}$$

En général, on utilise une version simplifiée de l'équation (1.29) en appliquant le complément de Schur sur le bloc supérieur à gauche, on obtient la version suivante de l'équation de mouvement sous contraintes :

$$\underbrace{\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\top\right)}_{\mathbf{A}} \boldsymbol{\lambda}^+ = \underbrace{-\left(h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} + \mathbf{J}\mathbf{v}\right)}_{\mathbf{b}} \tag{1.30}$$

On doit résoudre l'équation linaire $\mathbf{A}\boldsymbol{\lambda}^+ = \mathbf{b}$ afin de déterminer les impulsions $\boldsymbol{\lambda}^+ \in \mathbb{R}^m$. Ces impulsions nous permettent de déterminer les vitesses $\mathbf{v}^+$ des corps dans la prochaine pas du temps.

Avant de commencer la résolution, on doit tenir compte que certaines contraintes , tels que les constraintes de contact, nécessitent des conditions supplémentaires afin de rendre le problème plus réel en tenant compte d'autres phénomènes tels que la friction.

1.1.6.2 Problème de complémentarité linéaire (LCP)

Dans une simulation des corps rigides, on va trouver des contraintes bilatérales, mais également des contraintes unilatérales tels que les contacts. Les contacts, et les contraintes unilatérales en général, introduisent d'autres conditions de faisabilité et de complémentarité a l'équation (1.30). Les conditions de complémentarité et de faisabilité sont décrit par les équations (1.26).

En combinant les différentes équations décrivant le mouvement des corps rigides sous contraintes, on obtient un problème de complémentarité linéaire mixte (MLCP) formée par l'ensemble des équations suivantes :

$$\begin{aligned} &\mathbf{A}\boldsymbol{\lambda}^{+} - \mathbf{b} = \mathbf{w}^{+} = \mathbf{w}_u^{+} - \mathbf{w}_l^{+} \\ &0 \leq \mathbf{w}_l^{+} \perp \boldsymbol{\lambda} - \boldsymbol{\lambda}^{lo} \geq 0 \\ &0 \leq \mathbf{w}_u^{+} \perp \boldsymbol{\lambda}^{hi} - \boldsymbol{\lambda} \geq 0 \\ &0 \leq \mathbf{w}_u^{+} \perp \mathbf{w}_l^{+} \geq 0. \end{aligned} \tag{1.31}$$

Dans l'équation ci-dessus, $\mathbf{w}^{+}$ est la vitesse implicite de relâchement des constraintes. Notez également qu'on a $\mathbf{w} = 0$ pour les contraintes bilatérales et $\mathbf{w} = 0$ pour un contact statique. Sinon, les contacts peuvent se séparer ou glisser dans le cas où $\mathbf{w} \neq 0$.

1.1.6.3 Stabilisation des contraintes

Avant de commencer la résolution de notre système, on doit s'assurer que notre problème, formulé comme un problème de complémentarité linéaire mixte (MLCP), est bien conditionné et que les contraintes de mouvement ne sont pas violées à cause de l'accumulation des erreurs numériques. Cela permet d'avoir des simulations plus fluides avec la réduction des artefacts au niveau de la position et de la vitesse.

Baumgarte (1972) a proposé une approche pour résoudre le problème de violation de contraintes et Cline & Pai (2003) ont utilisé cette méthode dans la stabilisation des contraintes dans la simulation des corps rigides. Cette approche considère que la force d'une contrainte s'écrit sous la forme implicite d'une force de ressort. Alors, pour une contrainte i, on a l'expression de la force suivante :

$$\lambda_i^+ = -k_i\phi_i^+ - \beta_i w_i. \tag{1.32}$$

où ϕ_i est le terme de violation de contrainte i, w_i est la vitesse relative dans l'espace de contrainte, k_i la raideur du ressort et β_i l'amortissement du ressort.

Dans le cas ou toutes les contraintes sont stabilisées, on peut écrire l'équation des contraintes sous la forme

$$\mathbf{J}\mathbf{v}^+ + \mathbf{C}\boldsymbol{\lambda}^+ = -\gamma\frac{\boldsymbol{\phi}}{h}, \tag{1.33}$$

où $\mathbf{C} \in \mathbb{R}^{m\times m}$ est une matrice diagonale positive et $\gamma \in \mathbb{R}$ est un paramètre de réduction d'erreur. En général, on choisit une valeur proche de $\gamma = 1$.

Maintenant, en considérant la stabilisation des contraintes, on peut réécrire l'équation (1.29) de la façon suivante :

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}^\top \\ \mathbf{J} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{v}^+ \\ \boldsymbol{\lambda}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{M}\mathbf{v} + h\mathbf{f} \\ -\gamma\frac{\boldsymbol{\phi}}{h} \end{bmatrix} \tag{1.34}$$

Enfin, on peut présenter notre version définitive de la formulation MLCP du problème :

$$\begin{aligned} &\underbrace{\left(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\top + \mathbf{C}\right)}_{\mathbf{A}}\boldsymbol{\lambda}^+ - \underbrace{\left(-\gamma\frac{\boldsymbol{\phi}}{h} - h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} + \mathbf{J}\mathbf{v}\right)}_{\mathbf{b}} = \mathbf{w}^+ = \mathbf{w}_u^+ - \mathbf{w}_l^+ \\ &0 \le \mathbf{w}_u^+ \perp \boldsymbol{\lambda} - \boldsymbol{\lambda}^{lo} \ge 0 \\ &0 \le \mathbf{w}_l^+ \perp \boldsymbol{\lambda}^{hi} - \boldsymbol{\lambda} \ge 0 \\ &0 \le \mathbf{w}_u^+ \perp \mathbf{w}_l^+ \ge 0 \end{aligned} \tag{1.35}$$

1.2 Solveurs numériques

Dans cette section, on va s'intéresser à la résolution d'un problème MLCP définit par l'équation (1.31). Pour la résolution cette formulation, il y a une variété des méthodes numériques employées.

On va considérer dans cette section que $\mathbf{x}$ est le vecteur d'inconnues au lieu d'utiliser les impulsions de contraintes $\boldsymbol{\lambda}$.

1.2.1 Solveurs à point-fixe

Dans cette partie, on va se concentrer sur les solveurs à point fixe avec séparation et plus particulièrement sur la méthode PGS. Les méthodes de point fixe avec séparation sont les méthodes basées sur la décomposition de la matrice principale $\mathbf{A}$ sous la forme d'une différence entre de deux matrices $\mathbf{A} = \mathbf{Q} - \mathbf{N}$. Ces méthodes vont générer une expression de $\mathbf{x}^{(k+1)}$ dans l'itération $(k+1)$:

$$\mathbf{x}^{(k+1)} = \min(\mathbf{u}, \max(\mathbf{l}, \mathbf{Q}^{-1}(\mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}))). \tag{1.36}$$

On va noter les vecteurs $\mathbf{u}$ et $\mathbf{l}$ comme le vecteur de limite supérieure et le vecteur de limite inférieure respectivement.

En général, on a trois méthodes de décomposition qui se résument dans le tableau 1.1.

Tableau 1.1 Méthodes à point fixe avec les décompositions de la matrice $\mathbf{A}$ en considerant que : $\mathbf{D}$ est la matrice diagonale formée par les éléments diagonaux de $\mathbf{A}$, $\mathbf{L}$ est la matrice triangulaire inférieure stricte de $\mathbf{A}$ et $\mathbf{U}$ est la matrice triangulaire supérieure stricte de $\mathbf{A}$

Méthodes	**Q**	**-N**
Jacobi	$\mathbf{D}$	$\mathbf{L} + \mathbf{U}$
PGS	$\mathbf{L} + \mathbf{D}$	$\mathbf{U}$
PSOR	$\mathbf{D} + \alpha\mathbf{L}$	$(1-\alpha)\mathbf{D} - \alpha\mathbf{U}$

On va s'intéresser à la méthode du Gauss-Seidel dans les deux sections 1.2.1.1 et 1.2.1.3 avec une description plus détaillée de l'algorithme, ainsi que les techniques d'améliorations de ce type de solveur pour les simulations des corps rigides.

1.2.1.1 Gauss-Seidel Projetée

Nous divisons la matrice principale **A** dans notre problème en une matrice triangulaire strictement supérieure **U**, une matrice triangulaire strictement inférieure **L** et une matrice diagonale **D** :

$$\mathbf{A} = \mathbf{U} + \mathbf{L} + \mathbf{D} \tag{1.37}$$

Dans notre formulation LCP/MLCP, on réduit le système équations linéaires en supprimant les conditions de **x** et en mettant la variable de résidu à zéro ($\mathbf{w} = \mathbf{0}$).

$$(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} - \mathbf{b} = \mathbf{0} \tag{1.38}$$

La méthode de Gauss-Seidel performe des itérations sur des point fixes du système linéaire tel que :

$$\left(\mathbf{x}^{(k+1)}\right)_i = \left((\mathbf{L} + \mathbf{D})^{-1}\mathbf{b} - (\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}\mathbf{x}^{(k)}\right)_i \tag{1.39}$$

Aussi, on peut écrire cette méthode sous la forme d'un ensemble des équations scalaires :

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij}x_j^{(k)}}{A_{ii}} \tag{1.40}$$

où A_{ij} correspond à l'élément dans la i-ème ligne et la j-ème colonne de la matrice **A** et $x_i^{(k+1)}$ est le i-ème composant de $\mathbf{x}^{(k+1)}$.

La solution trouvée pour le système linéaire ($\mathbf{A}\mathbf{x} - \mathbf{b} = 0$), doit être projetée pour changer les éléments trouvés à l'extérieur des limites à l'intérieur des limites. Cela permet d'avoir des

solutions réalisables pour notre problème de complémentarité linéaire.

$$x_{\text{proj},i} = \max(\min(x_i, u_i), l_i) \tag{1.41}$$

Cette méthode de résolution de problème de complémentarité, bien que PGS ne vérifie pas la complémentarité de **w** et **x** et la faisabilité de **w** au cours de la résolution, est très utilisée. La popularité de cet algorithme, surtout dans le domaine de l'infographie, est due à la rapidité de l'algorithme à donner une solution approximative du problème. Le grand avantage de PGS est que nous n'avons pas besoin de factoriser ou d'inverser la matrice principale **A** ou sa sous-matrice, cependant on doit résoudre n équations scalaires linéaires.

On vérifie la convergence en calculant la variation de **x** obtenue en une itération $\delta\mathbf{x} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ et on le compare à une certaine tolérance donnée. L'algorithme est bien illustré dans 1.1.

Les entrées dans cet algorithme sont :

- **A** : Matrice principale du système.
- **b** : Vecteur cote droite du système à résoudre.
- **l** : Vecteur limite inférieure
- **u** : Vecteur limite supérieure
- $\mathbf{x}_0$: Vecteur de la solution initiale
- k_{max} : Nombre maximal d'itérations
- ϵ_{change} : Tolérance sur le changement entre deux itérations successives.
- ϵ_{error} : Tolérance sur l'erreur.

1.2.1.2 Calcul d'erreur

Le calcul d'erreur est l'un de facteur influenceur dans la production d'une meilleure convergence possible. Enzenhöfer, Andrews, Teichmann & Kövecses (2018) ont comparé des méthodes différentes de calcul de l'erreur de résolution de l'équation MLCP. Le calcul de notre erreur ne sera pas basé sur une solution de référence, mais plutôt sur les variables existant dans nos équations MLCP. Pour avoir des résultats comparables avec l'existant, on a utilisé ce qu'on

Algorithme 1.1 Algorithme PGS

Algorithme : Algorithme PGS
Input : $\mathbf{A}$, $\mathbf{b}$, $\mathbf{l}$, $\mathbf{u}$, $\mathbf{x}_0$, k_{max}, ϵ_{change}, ϵ_{error}
Output : $\mathbf{x}$

$n \leftarrow$ size($\mathbf{A}$)
$k \leftarrow 0$
$\mathbf{x} \leftarrow \mathbf{x}_0$
while $(k < k_{max})$ *et* $(\|\delta\mathbf{x}\| > \epsilon_{change})$ *et* $(e > \epsilon_{error})$ **do**
  for $i = 1 : n$ **do**
    $\Delta \leftarrow b_i$
    for $j = 1 : i - 1$ **do**
      $\Delta \leftarrow \Delta - A_{i,j}x_j$; /* éléments triangulaires inférieurs */
    end for
    for $j = i + 1 : n$ **do**
      $\Delta \leftarrow \Delta - A_{i,j}x_j$; /* éléments triangulaires supérieurs */
    end for
    $\Delta \leftarrow \frac{\Delta}{A_{i,i}}$; /* division par l'élément diagonal */
    $x_{new} \leftarrow \max\left(\min\left(\Delta, u_i\right), l_i\right)$
    $\delta x_i \leftarrow |x_{new} - x_i|$
    $x_i \leftarrow x_{new}$
  end for
  $\mathbf{w} \leftarrow \mathbf{Ax} - \mathbf{b}$
  $e \leftarrow$ **LCPError**$(\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u})$
  $k \leftarrow k + 1$
end while

appelle l'*erreur résiduelle naturelle*. Le calcul de cette erreur se fait comme

$$e = \|\boldsymbol{\psi}\|_1 = \sum_{i=1}^{m} |\boldsymbol{\psi}_i|, \tag{1.42}$$

avec $\boldsymbol{\psi} = [\boldsymbol{\psi}_1 ... \boldsymbol{\psi}_m]^\top$ le vecteur composé des erreurs de toutes les contraintes du système.

Le calcul de l'erreur de la contrainte i se définit par l'expression suivante qui extrait le maximum de violation des conditions de la contrainte :

$$\psi_i = \max\left(\min(u_i - x_i, w_i^-), \min(x_i - l_i, w_i^+)\right) \tag{1.43}$$

Dans l'algorithme 1.1, la fonction **LCPError** calcule l'erreur selon l'équation 1.42.

1.2.1.3 Les techniques d'améliorations

Plusieurs améliorations ont été appliquées sur les algorithmes itératifs pour améliorer la convergence et réduire le temps de calcul dans la résolution des problèmes des contraintes dans la simulation des corps rigides.

Erleben (2005) a utilisé une version par bloc de l'algorithme PGS dans la résolution des problèmes de complémentarités linéaires en boite (BLCP) pour avoir une convergence linéaire dans les simulations avec un ensemble structuré des corps rigides. Aussi, le même chercheur (Erleben, 2007) a introduit la propagation des chocs dans l'algorithme pour résoudre les problèmes de contacts formulée comme un problème de complémentarité basé sur la vitesse. Dans sa formulation l'auteur a considéré que la relations des points de contact peut être appliquée, à travers le temps, sur des morceaux spatiaux disjoints basés sur des couches le long de la direction de la gravité. Cette technique permet d'accélérer la propagation des chocs dans la direction de gravité, du haut vers le bas.

Parmi les autres techniques utilisées pour l'accélération de la convergence, on trouve la réorganisation des variables des contraintes à résoudre avec les méthodes itératives. Poulsen, Niebe & Erleben (2010) ont décrit différentes stratégies pour réorganiser les variables à résoudre dans les problèmes de contact avec frottement en utilisant la méthode PGS.

Parmi ses stratégies, on trouve les stratégies de réorganisations échelonnées qui séparent les variables en deux sous groupes à résoudre séparément, le groupe des variables des réactions normales et le groupe des variables des forces de frottements. Dans un autre travail plus récent, Andrews, Erleben, Kry & Teichmann (2017) ont cherché une stratégie pour trouver le meilleur ordre de résolution pour l'algorithme PGS dans des simulations structurées. Ils ont fait des recherches excessives sur des petites simulations dans le but de prouver l'existence d'une permutation idéale pour avoir une meilleure convergence que les stratégies déjà existantes.

La technique de réorganisation des variables était aussi appliquée aussi pour la résolution des contacts dans la simulation dynamique des corps mous. Fratarcangeli, Tibaldo & Pellacini (2016) ont proposé une version du PGS plus stable et plus rapide en termes de convergence pour ce genre de simulations. Ils ont créé une implémentation parallèle sur GPU du Gauss-Seidel en utilisant la coloration aléatoire des graphes des contraintes. Leur implémentation a donné des excellents résultats pour des systèmes avec des matrices larges et creuses.

Parmi les autres techniques utilisées, on trouve les techniques d'accélération basées sur l'impulsion du moment. Le travail de Wang (2015) avec l'utilisation des polynômes de Chebychev et le travail de Mazhar, Heyn, Negrut & Tasora (2015) avec l'utilisation de l'élan de Nesterov permet d'avoir une accélération de la convergence par rapport à l'implémentation standard de l'algorithme PGS.

Dans un autre article, Miyamoto & Yamashita (2021) proposent une méthode pour l'accélération de la résolution des problèmes de complémentarité linéaire servant dans la simulation interactive des corps rigides. L'amélioration de convergence est basée sur l'utilisation de «Accelerated Modulus-based Gauss–Seidel (AMGS)» en donnant une condition de convergence plus précise que la version classique de AMGS. La convergence de la méthode développée dans cet article est beaucoup meilleur que l'algorithme PGS mais pas en temps réel.

1.2.2 Solveurs directs

Les solveurs directs exploitent les conditions de complémentarité et de faisabilité du modèle en utilisant l'indexation des variables pour réduire la taille du système. Pour ce type de méthodes, également appelées *méthodes avec pivotement*, il existe plusieurs algorithmes développés avec différentes stratégies de pivotement :

- On trouve les algorithmes avec pivotement incrémentiel entre les variables serrées et les variables libres comme l'algorithme de Lemke (Lloyd, 2005).
- On trouve aussi les algorithmes avec pivotement par blocs tel que l'algorithme de Judice (Júdice & Pires, 1994).

De plus, plusieurs améliorations ont été développées sur ce type de méthodes, comme le travail proposé par Enzenhöfer, Lefebvre & Andrews (2019) où ils ont réussi à accélérer la version standard de l'algorithme de Judice jusqu'à trois fois en termes de temps de calcul. De même, le travail de Peiret, Andrews, Kövecses, Kry & Teichmann (2019) permet de paralléliser les méthodes de pivotement pour résoudre les problèmes de complémentarité linéaire en boîte (BLCP) pour la résolution des contacts dans les simulations de corps rigides.

1.2.3 Autres solveurs

La résolution des contraintes dans une simulation des corps rigides peut être réalisé en changeant la formulation et la manière de résolution. Silcowitz, Niebe & Erleben (2010) ont formulé les contacts sous la forme d'un problème de complémentarité linéaire, puis ils ont créé une combinaison entre la résolution avec pivotement et PGS. Il cherche l'ensemble des variables libres en faisant une résolution avec PGS. Ensuite, il utilise la méthode de gradient conjugue pour résoudre le système forme des contraintes libres. Après la résolution, on exclut l'ensemble des variables vérifiant les conditions de faisabilité ou de complémentarité dans la prochaine itération.

Les mêmes auteurs Silcowitz-Hansen, Niebe & Erleben (2010) ont proposé une autre approche itérative pour résoudre la formulation non-linéaire. Ils ont intégré la méthode gradient conjugué non linéaire et non lisse de type Fletcher-Reeves. Leur approche permet d'avoir des résultats acceptables en le comparant avec la résolution en utilisant PGS.

Erleben (2007) a proposé une méthode de fractionnement qui itère entre une résolution par pivotement de bloc pour le sous-ensemble combiné de contraintes d'articulation et de non-interpénétration, suivie d'une résolution itérative avec PGS pour les contraintes de non-interpénétration et de friction. Cette méthode génère des solutions précises pour les forces normales d'articulation et de contact tout en compromettant la précision des forces de friction. Les performances de cette méthode sont meilleures que les méthodes directes avec pivotement dans le cas où la précision des forces de friction n'est pas essentielle.

1.2.4 Synthèse

Les solveurs présentés ci-dessus sont des versions améliorées de solveurs standards déjà existants. Les techniques utilisées pour améliorer ces algorithmes sont variées et les objectifs sont également différents. Il existe des techniques utilisées pour augmenter la précision, des techniques pour améliorer le temps de calcul, des techniques pour améliorer le comportement de convergence, etc. Une des techniques les moins explorées dans le contexte d'amélioration de la convergence des solveurs à point fixe est le réglage des hyper-paramètres, en particulier pour l'algorithme PGS.

CHAPITRE 2

MÉTHODOLOGIE

Dans ce chapitre, on présente l'approche utilisée dans ce mémoire. Dans une première partie, on décrit les paramètres ajoutés dans l'algorithme PGS en détail. Ensuite, on introduit l'algorithme stratégies d'évolution avec adaptation de matrice de covariance. Enfin, on explique comment on utilise cet algorithme dans l'optimisation des paramètres introduit dans l'algorithme PGS.

2.1 Modifications du Gauss-Seidel Projetée

Dans cette section, on décrit les paramètres introduits dans l'algorithme PGS ainsi que les améliorations ajoutées pour avoir une implémentation de PGS plus performante.

2.1.1 Paramètres introduits

Afin d'avoir une meilleure convergence, on introduit un paramètre de régularisation $\kappa \in [0,\ 0.1]$ et un paramètre de relaxation $\alpha \in (0,\ 2]$ dans l'algorithme PGS. Ces deux paramètres sont des techniques standard dans le domaine de la simulation physique qui nécessitent un réglage des paramètres. D'ailleurs, Erleben (2007) propose d'utiliser la régularisation pour améliorer le conditionnement numérique du système résolu pour des simulations des corps rigides. En outre, Erleben (2004) utilise un facteur de relaxation pour accélérer la résolution des systèmes de contact dans les simulations physiques multicorps.

Le paramètre de régularisation permet d'améliorer le conditionnement du système linéaire à résoudre. Le conditionnement de notre système linéaire de contraintes est affecté par plusieurs facteurs tels que : la différence des masses entre les objets rigides interaction, la rigidité des contraintes, la redondance des contraintes au cours de la simulation, etc. Aussi, on note que la régularisation peut être reformulée comme une compliance et un amortissement des contraintes (Andrews & Erleben, 2021).

Le paramètre de relaxation permet d'accélérer ou décélérer la convergence en fonction du système résolu. Si le système est convergent, on a un α supérieure à 1.0 et si le système est divergent, on doit réduire α à une valeur inférieure à 1.0 afin de chercher une possibilité de convergence (Young, 1971).

L'introduction de ces deux paramètres va changer les expressions des variables scalaires (1.40) comme suit :

$$x_i^{(k+1)} = \alpha \frac{b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} x_j^{(k)}}{(A_{ii} + \kappa)} \tag{2.1}$$

La version utilisée pour l'optimisation de ces paramètres se trouve dans l'algorithme 2.1. Les entrées de cet algorithme sont les mêmes que celles de l'algorithme 1.1 avec l'ajout du paramètre de relaxation α et du paramètre de régularisation κ.

Algorithme 2.1 Version modifiée de l'algorithme PGS

Algorithme : Version modifiée de l'algorithme PGS
Input : **A**, **b**, **l**, **u**, $\mathbf{x}_0$, k_{max}, ϵ_{change}, ϵ_{error}, $\boxed{\alpha}$, $\boxed{\kappa}$
Output : **x**

$n \leftarrow$ size(**A**)
$k \leftarrow 0$
$\mathbf{x} \leftarrow \mathbf{x}_0$
while $(k < k_{max})$ *et* $(\|\delta\mathbf{x}\| > \epsilon_{change})$ *et* $(e > \epsilon_{error})$ **do**
    for $i = 1 : n$ **do**
        ⋮
        $\delta x_i \leftarrow \frac{\Delta}{A_{i,i}+\boxed{\kappa}}$; `/* les éléments diagonaux */`
        $x_{new} \leftarrow \max\left(\min\left(x_i + \boxed{\alpha}(\Delta - x_i), u_i\right), l_i\right)$
        ⋮
    end for
    ⋮
    $k \leftarrow k + 1$
end while

2.1.2 Version avec matrice creuse

Dans les simulations des corps rigides, les matrices principal dans la modélisation des contacts avec une formulation MLCP sont souvent des matrices creuses. On a rectifié la version du PGS pour tenir de cette propriété afin d'accélérer le temps de calcul en utilisant le travail proposé par Buluç, Fineman, Frigo, Gilbert & Leiserson (2009). Dans notre implémentation, on utilise le stockage *Compressed Row Storage (CRS)* qui génère l'ensemble des indices des valeurs non nulles dans la matrice ainsi que les valeurs correspondantes à ses indices (Buluç *et al.*, 2009). On peut observer les modifications réalisées sur PGS dans le pseudo-code de l'algorithme 2.2.

Dans cet algorithme, on trouve les mêmes entrées que dans l'algorithme 2.1 sauf qu'on remplace la matrice principale $\mathbf{A}$ par deux entrées : $\mathbf{A}^{inds}$, une matrice qui contient les indices des valeurs non nulles pour chaque ligne de la matrice $\mathbf{A}$, et $\mathbf{A}^{vals}$, une matrice qui contient les valeurs non nulles de la matrice $\mathbf{A}$. On note également que la première entrée de chaque ligne des tableaux $\mathbf{A}^{inds}$ et $\mathbf{A}^{vals}$ est l'entrée diagonale. Ce deux tableaux représentent le stockage *Compressed Row Storage (CRS)* de la matrice $\mathbf{A}$.

Les parcours de la matrice dans l'algorithme PGS sera plus optimale puisqu'on passe d'un élément à un autre sans passer par les zéros dans la matrice. Dans notre nouvelle implémentation, la mise à jour des variables de la solution et de l'erreur sera plus rapide que la version classique. On peut observer l'effet de cette modification dans la figure 2.1.

L'exécution de l'algorithme PGS est souvent utilisée au cours du processus d'optimisation. Donc, cette amélioration permet d'accroître les performances et d'obtenir les résultats de l'optimisation plus rapidement.

2.2 Stratégies d'évolution avec adaptation de matrice de covariance (CMA-ES)

Dans notre approche, on cherche une méthode pour choisir des valeurs qui permettent d'avoir la convergence à une tolérance donnée avec un minimum de nombre d'itérations possible. La formulation du problème ne permet pas de tester un simple algorithme d'optimisation puisque

Algorithme 2.2 PGS version avec matrice creuse

Input : $\boxed{\mathbf{A}^{inds}}$, $\boxed{\mathbf{A}^{vals}}$, $\mathbf{b}$, $\mathbf{l}$, $\mathbf{u}$, $\mathbf{x}_0$, k_{max}, ϵ_{change}, ϵ_{error}, α, κ
Output : $\mathbf{x}$

$sz \leftarrow \text{size}(\mathbf{A}^{vals})$
$k \leftarrow 0$
$\mathbf{x} \leftarrow \mathbf{x}_0$
while $(k < k_{max})$ *et* $(\|\delta\mathbf{x}\| > \epsilon_{change})$ *et* $(e > \epsilon_{error})$ **do**
  for $i = 1 : sz$ **do**
    $c \leftarrow \text{size}(A^{vals}(i,:))$
    $$\Delta \leftarrow \frac{b_i - \sum_{2 \le j \le c} \mathbf{A}^{vals}(i,j)\mathbf{x}\left(\mathbf{A}^{inds}(i,j)\right)}{\mathbf{A}^{vals}(i,1) + \kappa}$$
    $x_{new} \leftarrow \max(\min(x_i + \alpha(\Delta - x_i), u_i), l_i)$
    $\delta x_i \leftarrow |x_{new} - x_i|$
    $x_i \leftarrow x_{new}$
  end for
  /* Calcul de **w** */
  for $i = 1 : sz$ **do**
    $c \leftarrow \text{size}(A^{vals}(i,:))$
    $$w_i \leftarrow \sum_{1 \le j \le c} \mathbf{A}^{vals}(i,j)\mathbf{x}\left(\mathbf{A}^{inds}(i,j)\right) - b_i$$
  end for
  /* Calcul de l'erreur de résolution */
  $e \leftarrow \textbf{LCPError}(\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u})$
  $k \leftarrow k + 1$
end while

nous résolvons un problème d'optimisation non-linéaire ou les variables sont affectés les uns par les autres et un petit changement permet de changer le comportement de tout le système.

On a choisi un des plus robustes (Hansen, Auger, Ros, Finck & Pošík, 2010) algorithmes d'optimisation en une boite noire pour ces raisons, une stratégie d'évolution avec adaptation de la matrice de covariance. On utilise la méthode de *Covariance Matrix Adaptation Evolution Strategies* (CMA-ES) (Hansen, Müller & Koumoutsakos, 2003). En plus des raisons mentionnées

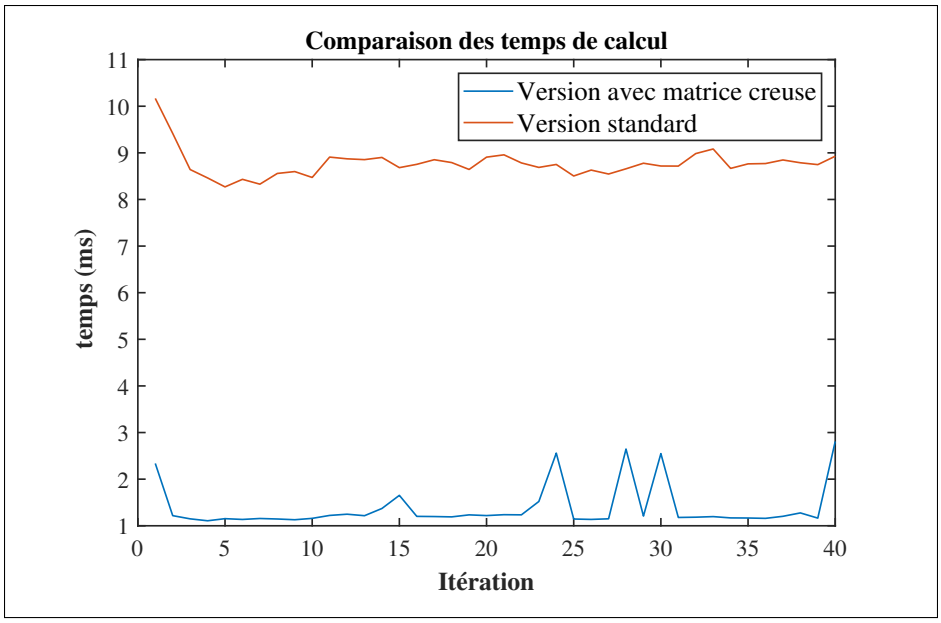


Figure 2.1 Comparaison du temps de calcul entre la version standard du PGS et la version avec matrice creuse

au-dessus, la section 2.3 présente une analyse profonde de la convergence de PGS en fonction des changements des variations des paramètres introduits dans notre solveur.

Dans cette section, on considère $\mathbf{x}$ le vecteur de paramètre à optimiser.

2.2.1 Définition

CMA-ES est une méthode d'optimisation numérique développée par Hansen *et al.* (2003). Cet algorithme est classé dans la famille des algorithmes évolutifs qui sont des méthodes stochastiques, non dérivées, qui sont souvent utilisées pour résoudre des problèmes d'optimisation numérique continus, non linéaires ou non convexes.

CMA-ES est un algorithme d'optimisation stochastique dans une boite noire. Pour ce type d'algorithme, on cherche à minimiser une fonction de fitness $f(\mathbf{x}) \in \mathbb{R}$ avec $\mathbf{x} \in \mathbb{R}^l$ est le vecteur des candidats à tester, avec l est le nombre des paramètres à optimiser. Dans un algorithme d'optimisation stochastique dans une boîte noire, on génère d'abord des échantillons

de paramètres en utilisant une probabilité de distribution donnée. Ensuite, on évalue les échantillons générés à l'aide de notre fonction de fitness. En tenant compte des résultats de l'évaluation dans une génération, on met à jour les valeurs des paramètres. On répète ce processus pendant plusieurs générations, et on s'arrête lorsque les critères d'arrêt choisis sont valides.

Toutes les étapes d'un algorithme d'optimisation stochastique dans une boite noire se résume dans l'algorithme 2.3.

Algorithme 2.3 Optimisation aléatoire dans une boite noire de f

```
1  Algorithme : Optimisation aléatoire dans une boite noire de f

2  Initialisation de la distribution des paramètres ζ
3  for génération g = 1,2,... do
4  |  Generer υ points independants de la distribution P(x|ζ^(g)) → x_1, ..., x_υ
5  |  Évaluer les échantillions x_1 ... x_υ
6  |  Mettre à jour les parameters ζ^(g+1)
7  |  if Les critères d'arrets sont valides then
   |  |  /* Voir la section 2.6.2                                    */
8  |  |  break
9  |  end if
10 end for
```

La distribution des paramètres utilisée dans CMA-ES pour générer les échantillons des points indépendants est une distribution normale à plusieurs variables.

2.2.2 Motivation du choix de l'algorithme

CMA-ES est un algorithme puissant dans de nombreux types d'applications et est considéré comme utile pour les problèmes ayant des objectifs très complexes et un grand nombre de variables à optimiser. Ce choix était basé sur l'avenir de ce projet puisqu'on voulait, dans le futur, inclure plus des paramètres dans notre solveur PGS.

Aussi, la particularité de CMA-ES par rapport aux autres algorithmes d'optimisation est que cet algorithme fonctionne très bien pour les fonctions objectives non linéaires, non convexes, non

séparables et mal conditionnés. Dans notre cas, nous avons une fonction de fitness non linéaire, non différentiable et avec des variables non séparables (voir la section 2.4).

De plus, une étude des optimisations de la boîte noire (Hansen *et al.*, 2010) a montré que cet algorithme est l'état de l'art par rapport aux autres algorithmes d'optimisation stochastique existants, avec de bonnes performances pour les fonctions difficiles ou pour le cas où les espaces de recherche sont de haute dimension.

2.2.3 Le fonctionnement de l'algorithme

Pour l'algorithme CMA-ES (Hansen *et al.*, 2003), le fonctionnement de l'algorithme se résume en quatre étapes principales : échantillonnage d'une nouvelle population de points de recherche, sélection et recombinaison, contrôle de la taille du pas et adaptation de la matrice de covariance.

D'abord, on génére une population de nouveaux points de recherche à partir d'une distribution normale multivariable pour chaque point, on évalue. Ensuite, on sélectionne les meilleurs points de la population actuelle pour mettre à jour la moyenne de la distribution de recherche. Le choix des points se fait à l'aide de la comparaison des sorties de la fonction de fitness. L'étape suivante sera la mise à jour du pas de contrôle/l'écart type de la distribution de recherche. La dernière étape, pendant une génération, est d'adapter la matrice de covariance pour réduire la zone de recherche. Ensuite, l'algorithme génère une nouvelle population jusqu'à l'un des différents critères d'arrêts est vérifiée.

Une description plus détaillée de l'algorithme est présentée dans l'annexe I.

2.3 Analyse de la convergence de la méthode PGS

Cette partie est dédiée pour l'analyser la convergence du PGS après l'introduction des paramètres dans l'algorithme afin de déduire la fonction de fitness convenable pour l'optimisation. Les courbes, générées pour cette section, sont le résultat des multiples résolutions des systèmes de la simulation de boite de marbre (voir section 3.1).

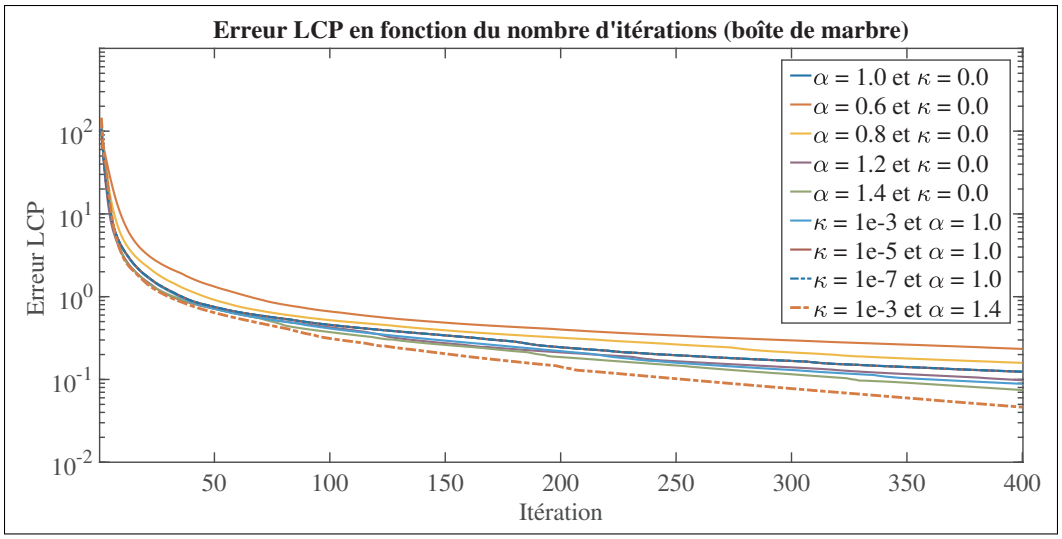


Figure 2.2 Erreur en fonction du nombre d'itérations dans une simulation avec variation des paramètres introduits (Stagnation)

Les courbes dans la figure 2.2 sont la représentation des différentes variations de la résolution d'un système linéaire avec la version modifiée du PGS. Dans les courbes, on peut observer l'existence d'une meilleure combinaison des paramètres afin de réduire l'erreur avec un nombre minimale d'itérations. Une petite variation dans l'un des paramètres peut donner une courbe de convergence différente selon le conditionnement du système linéaire, les vecteurs limites données, etc.

La fonction de fitness doit contenir l'effet de l'augmentation de nombre d'itérations et l'augmentation de l'erreur avec des pondérations bien précises. Aussi, l'ajout d'une grande quantité de régularisation ne sera pas bénéfique pour avoir une bonne simulation puisque la résolution sera loin du cas réel. Par conséquent, notre fonction de fitness doit tenir compte de la quantité de régularisation ajoutée.

Si on regarde la figure 2.3, on peut observer que même si on a une convergence dans le cas par défaut, le choix convenable de valeurs de α et de κ changent la convergence d'une manière significative. Ce changement prouve la sensibilité du choix de ces paramètres et leur effet sur la convergence de l'algorithme PGS.

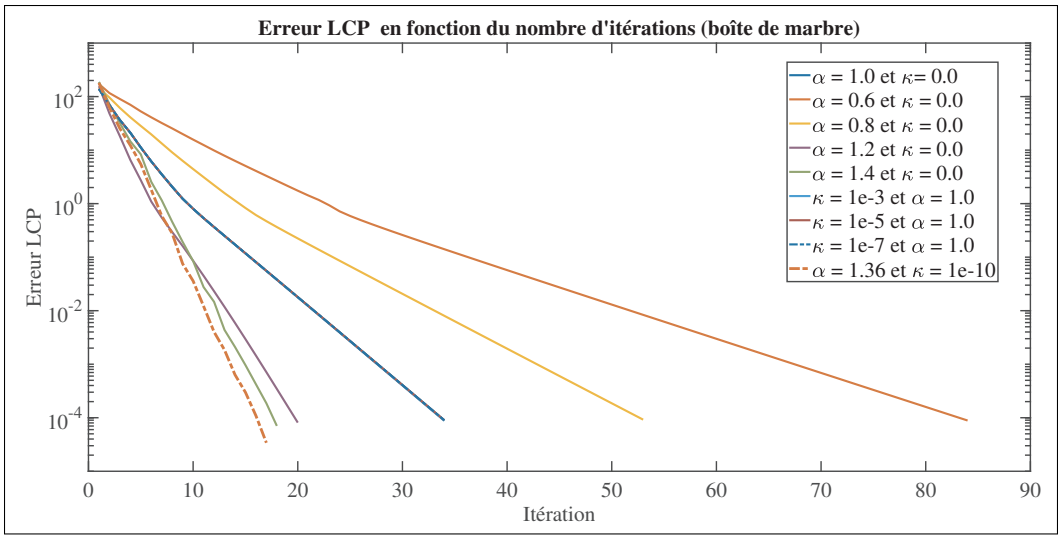


Figure 2.3 Erreur en fonction du nombre d'itérations dans une simulation avec variation des paramètres introduits (Forte convergence)

2.4 Définition de la fonction de fitness

Pour utiliser l'algorithme CMA-ES dans l'optimisation de différents paramètres, on doit concevoir une fonction de fitness qui permet d'optimiser le taux de convergence de l'algorithme PGS. Cette fonction doit tenir compte de plusieurs facteurs : la quantité de régularisation ajoutée, le nombre d'itérations et l'erreur au moment d'arrêt du solveur.

Dans notre cas, l'optimisation de ces facteurs doit se faire de manière indissociable en raison de la dépendance d'un facteur à l'autre. Par exemple, si on réduit le nombre d'itérations, on n'est pas sûrs d'avoir la tolérance cherchée pour notre solution. Plusieurs possibilités sont ouvertes pour la formulation de la fonction de fitness, comme l'idée d'utiliser une optimisation multi-objectifs. Cependant, comme première intuition, on a opté pour une fonction de fitness définie comme la somme de trois fonctions différentes :

- **Objective 1 :**

  La figure 2.4 montre une illustration de la convergence de deux résolutions différentes. Les deux résolutions convergent vers la même erreur, mais avec un nombre d'itérations différent. Pour les comparer, on considère seulement le nombre d'itérations. Dans notre

cas, on compare une résolution, réalisée à partir d'une combinaison de paramètres, avec la résolution présentant la meilleure convergence possible, c'est-à-dire le cas où le nombre d'itérations est égal à 1.

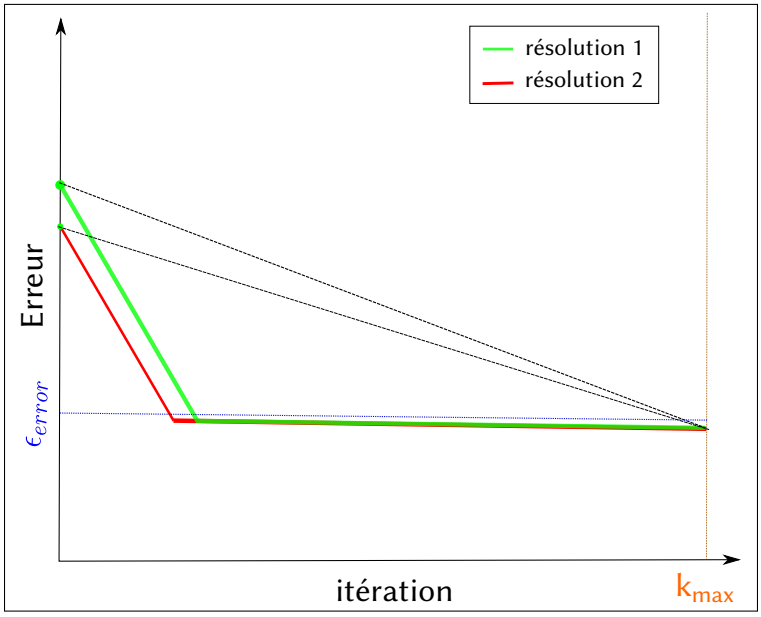


Figure 2.4 Convergence vers la même erreur

L'objectif 1 sera donc une fonction qui définit la différence entre le nombre d'itérations pour une résolution et 1 sans oublier de normaliser cette différence en la divisant par $(k_{max} - 1)$. La fonction de l'objective un s'écrit sous la forme de l'équation 2.2.

$$f_1 = \frac{k - 1}{k_{max} - 1} \tag{2.2}$$

avec,

- k est le nombre d'itérations de PGS.
- k_{max} est le nombre maximal d'itérations qu'on ne doit pas dépasser dans l'exécution de PGS.

- **Objective 2 :**

L'utilisation de la régularisation dans le système est dangereux puisque les contraintes du système seront modifiées et on se tourne vers la résolution d'un système légèrement différent du système initiale.

L'objective 2 sera donc une fonction qui pénalise l'ajout de régularisation dans notre système à résoudre. L'ajout de l'effet de la régularisation dans la fonction de fitness se fait à travers l'ordre de régularisation, c.-à-d. on a utilisé la différence entre $\log_{10}$ de la régularisation ajoutée et $\log_{10}$ de la régularisation maximale. Il convient également de mentionner que ce terme ne devrait pas être dominant dans la fonction de fitness globale, car l'ajout d'une grande quantité de régularisation modifie radicalement le système physique initial, ce qui affecte le réalisme de la simulation.

On définit cette fonction par l'équation 2.3.

$$f_2 = \frac{\log_{10}(\kappa) - \log_{10}(\kappa_{min})}{-1 - \log_{10}(\kappa_{min})} \tag{2.3}$$

avec,

- κ_{min} est égale à 10^{-16}.
- κ est la valeur de régularisation choisie pour tester.

- **Objective 3 :**

  Dans le cas où il n'y a pas de convergence, on compare les erreurs et on choisit la résolution dont l'erreur est la plus proche de la tolérance. La figure 2.5 est une illustration de cette situation. L'objective 3 sera une fonction qui tient compte de l'ordre de l'erreur atteinte avant l'arrêt de l'algorithme. La fonction d'objective 3 permet de pénaliser l'ordre d'erreur en ajoutant la différence entre l'ordre d'erreur acceptée et l'ordre d'erreur obtenue.

  La fonction de l'objective trois s'écrit sous la forme de l'équation 2.4.

$$f_3 = \log_{10}(\frac{e_{min}}{\epsilon_{error}}) \tag{2.4}$$

avec,

- e_{min} est la valeur minimale dans le vecteur des erreurs stockées à chaque itération du PGS.

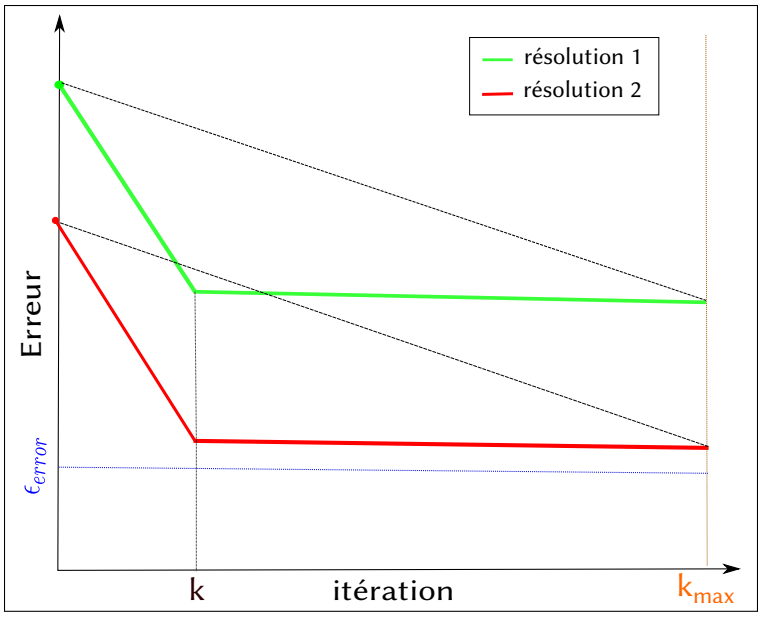


Figure 2.5 Convergence avec la même pente

- ϵ_{error} est la tolérance sur l'erreur (10^{-4}).

Il convient de mentionner que la valeur de la fonction f_3 sera positive ou nulle puisque si le solveur atteint une valeur inférieure ou égale à la tolérance ϵ_{error}, on considère que e_{min} est égal à ϵ_{error}.

On peut donc écrire la fonction de fitness comme une somme pondérés de ces trois fonctions sous la forme de l'équation 2.5.

$$f = s_1 f_1 + s_2 f_2 + s_3 f_3 \tag{2.5}$$

On remarque que toutes les fonctions f_1, f_2 et f_3 sont normalisées pour que leurs valeurs soient entre 0 et 1. Les valeurs de ses fonctions seront toujours dans l'intervalle $[0, 1]$ et l'effet de chaque fonction sera déterminée par les valeurs des poids s_1, s_2 et s_3. Les valeurs des poids s_1, s_2 et s_3, utilisées dans nos expériences, sont respectivement : 1.0, 0.5 et 1.2 pour toutes les simulations utilisées dans nos expérimentations. Ces valeurs sont choisies par propre intuition

par rapport aux systèmes simulés et après de multiples tests d'optimisation sur plusieurs scènes extraites de plusieurs simulations.

2.5 Évaluation de la fonction de fitness

Afin de régler et d'évaluer la fonction de fitness, on a utilisé la visualisation des différentes courbes. On a choisi quelques images à partir des simulations, décrites dans la section 3.1, pour évaluer le choix de la fonction de coût. Dans la figure 2.6, on a un ensemble des courbes permettent de visualiser l'évolution de la fonction ainsi que l'évolution des hyper paramètres et les axes principaux de la matrice de covariance. Dans les courbes en haut à gauche, on peut voir

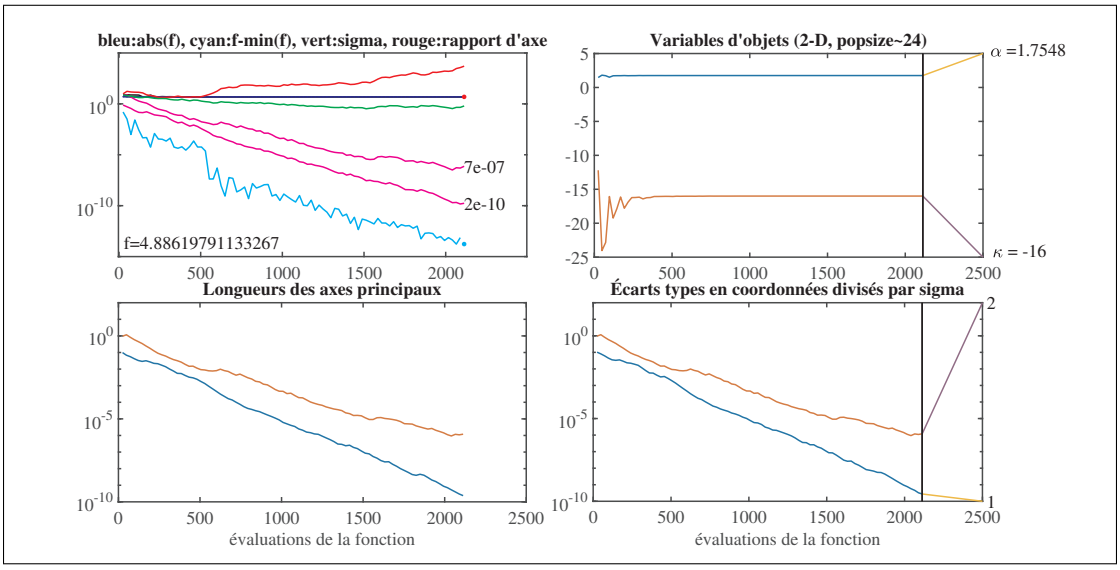


Figure 2.6 Évaluation de la fonction de fitness

la valeur absolue de la fonction de fitness (couleur bleu-ciel) qui décroît progressivement d'une évaluation à une autre. Cela permet de confirmer que le choix de la fonction de fitness est bien convenable.

Les courbes à droite en haut permettent de suivre l'évolution des paramètres en cours des différentes générations. Les courbes en bas à gauche permet de voir les valeurs des axes principaux de la zone de recherche, ce qui donne un aperçu sur la zone de recherche qui rétrécit progressivement d'une génération à une autre.

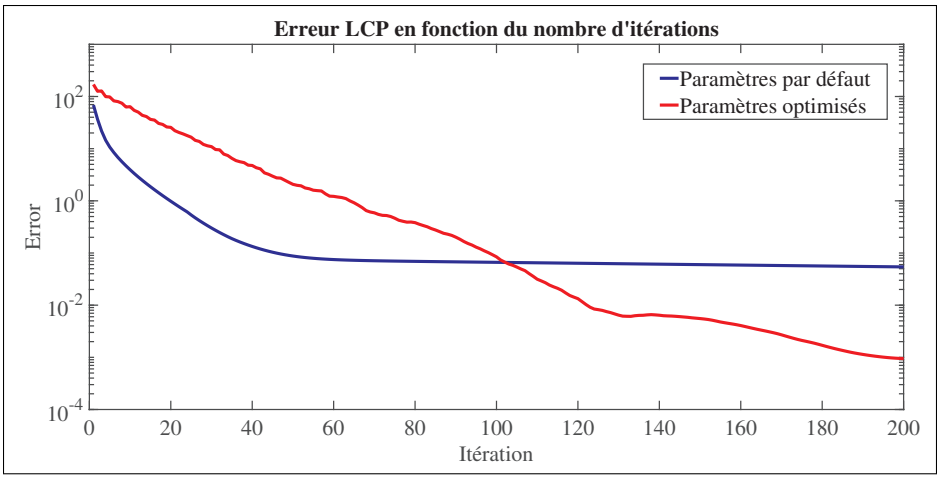


Figure 2.7 Stagnation de l'erreur sans l'utilisation des paramètres optimisés

Afin de valider la fonction de fitness, on a testé l'optimisation sur plusieurs images extraites à partir de plusieurs simulations. La figure 2.7 démontre l'effet de l'optimisation sur la convergence du PGS. On remarque que dans la version par défaut, on a une stagnation de l'erreur alors que dans le cas avec des paramètres bien choisis où on a une convergence vers une erreur avec l'ordre d'erreur cherché.

2.6 Réglages des paramètres du CMA-ES

Dans cette partie, on définit les paramètres nécessaires pour avoir le bon fonctionnement de l'algorithme. On distingue ici entre deux types de paramètres : paramètres d'entrées et paramètres précisant les critères d'arrêts.

2.6.1 Choix des paramètres d'entrées

Les paramètres d'entrées sont les variables fixées par l'utilisateur qui permet d'optimiser les hyper paramètres de l'algorithme PGS. Il y a deux paramètres à donner comme entrée à l'algorithme d'optimisation : Le vecteur de la moyenne et le vecteur de variance.

Dans notre cas, les deux vecteurs ont deux dimensions. Le vecteur moyen est initialisé par la valeur par défaut du paramètre de relaxation 1.0 et par le point de milieu de l'intervalle de l'ordre de paramètre de régularisation qui sera égale à (-8).

On suppose que les valeurs de κ seront dans l'ensemble $\{0\} \cup (10^{-16},\ 10^{-0.1}]$ en considérant que si l'ordre choisi est (-16), on réinitialise le paramètre de régularisation κ à zéro. Donc, on a un ordre de paramètre de régularisation entre (-16) et (-0.1).

En plus de ce deux paramètres, on trouve d'autres options incluses dans l'implémentation existante du CMA-ES tel que l'évaluation en parallèle et l'option de lancer plusieurs fois la recherche des points optimaux avec l'option d'augmenter la taille de la population à chaque redémarrage.

Dans notre cas, on a utilisé les évaluations en parallèle sans l'utilisation de l'option de redémarrage puisqu'une seule recherche sera suffisante pour notre cas.

2.6.2 Les critères d'arrêts

Afin d'avoir un point d'arrêt qui minimise la fonction de fitness, on doit bien fixer les critères d'arrêts. Ces critères, s'ils sont bien fixés, permettent de rendre l'exécution de cet algorithme d'optimisation efficace en minimisant des recherches ayant peu d'impact sur le résultat final. Pour l'algorithme CMA-ES, il existe plusieurs critères d'arrêts qui se classifient en deux groupes : les critères dépendant du problème et les critères stabilités numériques.

- **Les critères dépendant du problème :**
  - La tolérance de la fonction de fitness (*TolFun*) : cette valeur permet de s'arrêter à la meilleure valeur par rapport aux valeurs trouvées dans toutes les n_g générations précédentes. On a fixé cette valeur à 10^{-9}.
  - La tolérance des paramètres (*TolX*) : cette valeur permet de s'arrêter quand on atteint la précision cherchée pour nos paramètres à optimiser. On a fixé cette valeur à 10^{-12}.
- **Les critères stabilités numériques :** Il y a une variété des paramètres introduit dans l'algorithme de CMA-ES qui permet d'arrêter l'exécution lorsque la solution est numériquement

impossible de trouver tel que : la stagnation, conditionnement de la matrice de covariance, tolérance sur le changement de l'écart-type de la distribution normale, etc.

Le choix des critères d'arrêt se fait par l'expérimentation de nombreuses simulations choisies et après plusieurs essais d'optimisation sur plusieurs scènes extraites de plusieurs simulations.

CHAPITRE 3

RÉSULTATS ET DISCUSSIONS

Dans ce chapitre, on présente les résultats obtenus avec les paramètres optimisés en utilisant CMA-ES. Ces résultats seront analysés profondément pour détecter les relations et les règles de choix des valeurs appropriées des paramètres introduits dans PGS. Les tests effectués ont permis d'avoir un parcours clair pour l'optimisation et le choix des meilleures valeurs pour les paramètres introduits dans notre solveur de contraintes de mouvement.

3.1 Simulations et collecte des données

Dans notre implémentation, on a utilisé le moteur des jeux Vortex de CM Labs (CM Labs Simulations, 2019) pour créer les simulations et générer les données numériques. Tous les éléments des simulations testées sont sauvegardés sous la forme des fichiers "HDF5" (Koranne, 2010).

Les informations sur le système dynamique ainsi que les conditions de simulation et les informations à propos la collision sont organisées sous la forme des sous dossiers dans ces fichiers "HDF5" pour une succession des scènes pour toutes les simulations. L'enregistrement de ces données était réalisé en utilisant un plugin implémenté en C++. L'implémentation des différentes versions du solveur PGS ainsi que l'algorithme d'optimisation sont écrites en MATLAB.

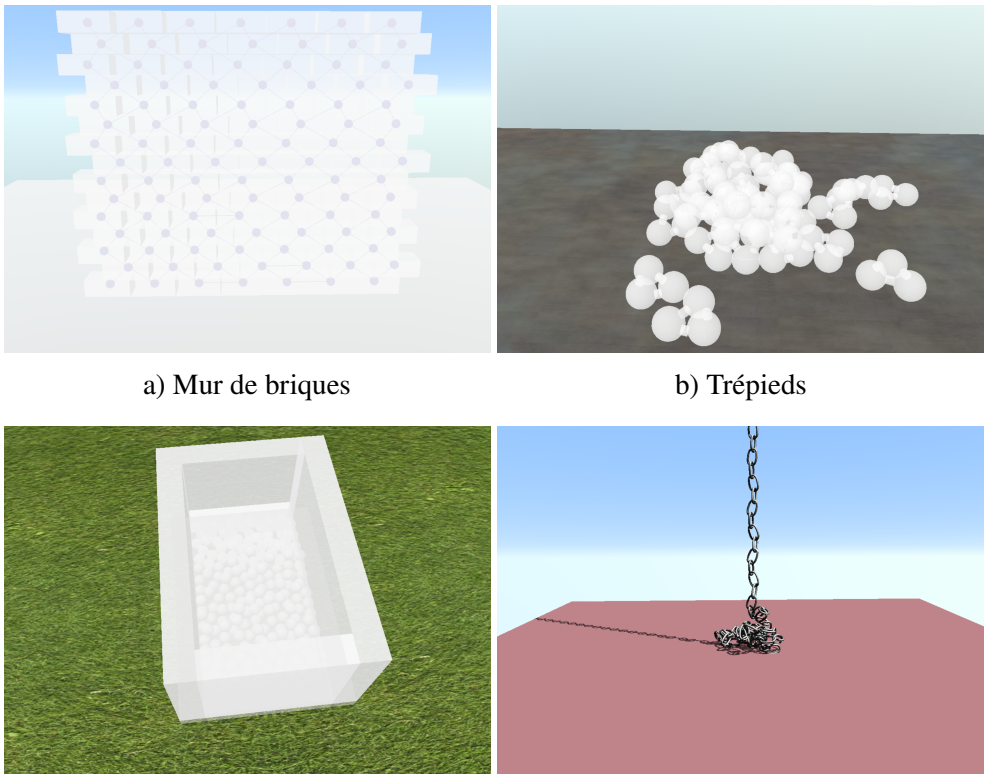
Afin d'avoir une comparaison convenable entre les résultats d'optimisation pour les quatre simulations, on fixe les valeurs de certains paramètres :

- Coefficient de frottement : le coefficient de frottement entre les corps rigides dans une simulation $\mu = 1.0$.
- Pas du temps[s] : La durée temporelle entre les images d'une simulation $h = \frac{1}{60}$ (secondes)
- Nombre maximal d'itérations : C'est le nombre maximal d'itérations à n'est pas dépassé $k_{max} = 200$. Ce choix d'un nombre d'itérations maximal est fait pour réaliser une comparaison

équitable entre les simulations afin d'avoir des interprétations cohérentes pour toutes les simulations sans dépendance de cette valeur.

- Tolérance de l'erreur : C'est la tolérance sur l'erreur de résolution (erreur de résolution naturelle). On l'a fixé à cette valeur $\epsilon_{error} = 10^{-4}$.
- Tolérance de changement : C'est la tolérance sur les changements de la solution entre les itérations. On l'a fixé à cette valeur $\epsilon_{change} = 10^{-9}$.

3.1.1 Exemples



a) Mur de briques b) Trépieds

c) Boîte de marbre d) Chute d'une chaîne

Figure 3.1 Simulations utilisées pour générer les données numériques

Dans cette section, on présente les simulations utilisées pour générer les données numériques utilisées pour l'optimisation des paramètres. La figure 3.1 montre les quatre simulations choisies : Mur de briques, trépieds, boîte de marbre et chute de chaîne.

La simulation du mur de briques contient plusieurs contacts sans mouvement introduit pendant la simulation. Ce type de simulation permet de voir la stabilité du solveur sans intervention ou introduction de forces sur les corps en mouvement pendant une simulation. Dans la simulation de trépieds, on peut voir un nombre limité de contacts générés pendant la simulation. La simulation de la boite de marbre est une extension de la simulation de trépieds avec beaucoup plus de contacts créés au cours du temps. La dernière simulation (Chute d'une chaîne) contient une combinaison de contraintes bilatérales et unilatérales qui nous permet de voir l'effet de l'introduction de contraintes bilatérales dans la simulation qui n'étaient pas présentes dans les autres simulations.

Tableau 3.1 Nombre des contraintes dans les simulations

Simulations	**Contraintes bilatérales**	**Contraintes unilatérales**	**Total**
Mur de briques	0	771 – 832	771 – 832
Trépieds	0	7 – 62	7 – 62
Boite de marbre	0	7 – 479	7 – 479
Chute d'une chaîne	99	0 – 101	99 – 200

Le tableau 3.1 résume le nombre et les types de contraintes présentes dans chaque simulation. Ce tableau permet de visualiser l'effet du type de contraintes et du nombre de contraintes sur les résultats de l'optimisation.

3.1.1.1 Mur de briques

Dans cette simulation, on a 112 briques ayant une masse de 1.0 Kg. Les briques sont organisées d'une manière dense et structurée sans avoir des espaces entre eux. L'absence des petits écarts entre les briques permet la génération des contacts dès le début de la simulation.

3.1.1.2 Trépieds

Dans cette simulation, on a généré les données à partir de 28 trépieds. Chaque trépied est formé de trois sphères liées par trois cylindres avec une masse égale à 3.5 Kg. Les trépieds tombent sur une terre fixe et créent une incrémentation des contacts au cours du temps.

3.1.1.3 Boîte de marbre

Dans cette simulation, on a 200 balles de 1.0 Kg et de rayon 0.25 m tombant dans une boite. Les contacts sont créés de manière incrémentale entre les balles et entre les côtés formant la boite.

3.1.1.4 Chute d'une chaîne

Dans cette simulation, nous allons simuler la chute d'une chaîne métallique sur un sol rigide. La chaîne est constituée de 100 maillons reliés entre eux où chaque maillon pèse 1.0 Kg. En plus des contraintes de liaisons entre les maillons, il y a une génération de contact incrémentielle entre les maillons eux-mêmes et les maillons et le sol.

3.2 Les résultats d'optimisation

Les figures 3.2 et 3.3 représentent respectivement l'évolution de la valeur optimale des paramètres de relaxation et de régularisation à chaque image pour les quatre simulations.

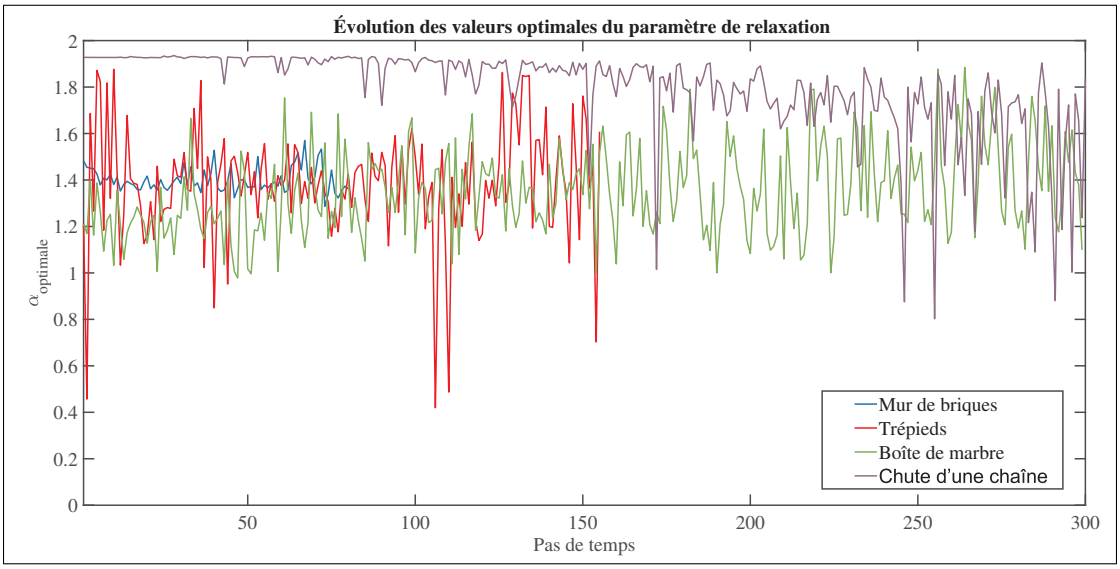


Figure 3.2 Valeurs de relaxation optimales au cours des simulations

On peut voir plus de variance pour le cas de la boîte de marbre. Cela pourrait être dû au fait que plusieurs contacts ont été générés en même temps, ce qui entraîne probablement une redondance des contraintes. Aussi, lorsque on observe la courbe pour la simulation de la chute d'une chaîne, nous remarquons que la variance du paramètre de relaxation est plus grande de manière décroissante. Ceci est dû à l'augmentation de la complexité du problème avec l'augmentation du nombre de contraintes au cours du temps.

De même pour la régularisation, les valeurs choisies sont plus élevées dans le cas où les systèmes sont mal conditionnés comme dans le cas de la boîte de marbre. Dans cette simulation, il y a création d'un nombre de contraintes plus important au cours du temps que dans la simulation des trépieds.

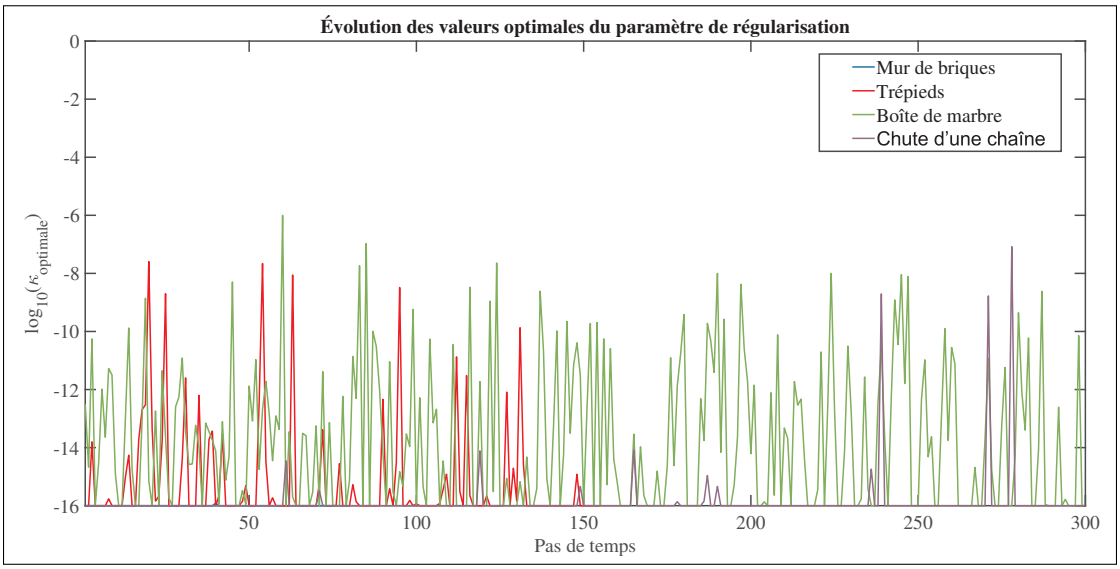


Figure 3.3 Valeurs de régularisation optimales au cours des simulations

Les tableaux 3.2 et 3.3 résument les valeurs moyennes avec les écart-types des valeurs des paramètres optimisées pour les quatre simulations présentées précédemment.

Il est nécessaire de mentionner que ces résultats d'optimisation ont pris beaucoup de temps pour les générer, car il est faut tester des milliers d'évaluations des combinaisons possibles des paramètres pour chaque scène, puis il est nécessaire de répéter ces optimisations pour un certain

Tableau 3.2 Valeurs de la moyenne et de l'écart-type du paramètre de relaxation optimal

Simulations	Moyenne	Écart-type
Mur de briques	1.40	0.05
Trépieds	1.41	0.27
Boite de marbre	1.49	0.26
Chute d'une chaîne	1.80	0.19

Tableau 3.3 Valeurs de la moyenne et de l'écart-type du paramètre de régularisation optimal

Simulations	Moyenne	Écart-type
Mur de briques	0.00	2.52e-18
Trépieds	2.83e-10	2.35e-09
Boite de marbre	2.07e-09	3.97e-08
Chute d'une chaîne	2.87e-10	4.77e-09

nombre d'images d'une simulation donnée. En général, pour une scène, le temps d'optimisation est entre 30 s et 15 minutes pour obtenir des résultats en fonction de la taille et de la complexité du problème que nous résolvons. Par exemple, une scène de la boite du marbre nécessite en moyenne environ 15 minutes alors qu'une scène de la simulation des trépieds nécessite moins qu'une minute pour générer les résultats. Le long temps de calcul est dû aux multiples tests des paramètres, puis à la répétition de la résolution d'un système linéaire à l'aide de PGS, chaque fois que CMA-ES choisi de nouvelles combinaisons de paramètres à évaluer.

3.3 Analyse des résultats

Dans cette section, on commence par une description des variables de système, puis une analyse des relations de ces variables avec les paramètres optimisées.

3.3.1 Choix des variables du système

Le choix de ces variables est basé sur une analyse des propriétés physique des simulations créées. On affiche l'évolution du paramètre de relaxation et du paramètre de régularisation en fonction de quatre variables différents :

- **Conditionnement :** le conditionnement de la matrice $\mathbf{A}$, ou $\text{Cond}(\mathbf{A}) = \frac{|\lambda_{\max}(\mathbf{A})|}{|\lambda_{\min}(\mathbf{A})|}$, avec $\lambda_{\max}$ et $\lambda_{\min}$ sont respectivement les valeurs propres maximale et minimale de la matrice $\mathbf{A}$. Cette valeur permet de donner une idée à propos le conditionnement du système linéaire : Si on a un indice de conditionnement est élevée, le système est dit mal conditionnée et si on a un indice de conditionnement est faible, le système est dit bien conditionnée.
- **Nombre d'itérations :** Le nombre d'itérations nécessaire pour avoir la convergence vers la tolérance cherchée permet de donner une idée sur la rapidité de convergence de l'algorithme PGS.
- **La taille du système :** La taille du système permet d'évaluer la variance des paramètres introduits en fonction la variance de la taille du système résolus.
- **La norme du vecteur b :** la norme du vecteur $\mathbf{b} = -\left(h\mathbf{J}\mathbf{M}^{-1}\mathbf{f} + \mathbf{J}\mathbf{v}\right)$ est une indication sur l'énergie cinétique du système. Tracer l'évolution des choix des paramètres optimaux en fonction de l'énergie cinétique pourrait donner une explication physique des valeurs optimales trouvées. L'unité de la norme de $\mathbf{b}$ est le *Joule (J)*.

3.3.2 Optimisation du paramètre de relaxation

Dans cette section, on présente les résultats d'optimisation du paramètre de relaxation α. On fait notre analyse sur les variations des valeurs optimisées du paramètre de relaxation en fonction des variables du système pour toutes les simulations.

3.3.2.1 Conditionnement

Si on regarde la figure 3.4 représentant le paramètre de relaxation en fonction du $\log_{10}$ du conditionnement du système linéaire, on peut approximer les valeurs de relaxation optimale avec

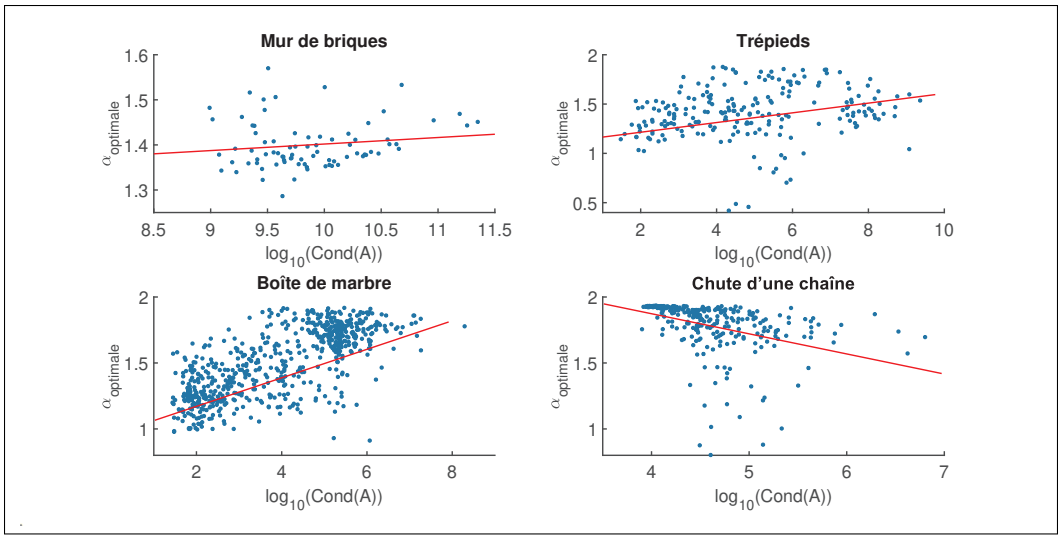


Figure 3.4 Valeurs de relaxation optimales en fonction du $\log_{10}$ du conditionnement de **A**

des relations linéaires en fonction du $\log_{10}(\text{Cond}(\mathbf{A}))$. Ces relations s'écrivent sous la forme de l'équation 3.1 où a et b sont de paramètres à préciser pour chaque simulation (voir tableau 3.4).

$$\alpha_{\text{optimale}} \approx a \log_{10}(\text{Cond}(\mathbf{A})) + b \tag{3.1}$$

On remarque que la pente de l'équation 3.1 pour la chute d'une chaîne est négative. Dans cette simulation, on a ajouté des contraintes bilatérales. Cette condition peut expliquer la diminution de la valeur du paramètre de relaxation avec l'augmentation du conditionnement du système à résoudre.

Tableau 3.4 Valeurs des relations linéaires entre α_{optimale} et $\log_{10}(\text{Cond}(\mathbf{A}))$

Simulations	a	b
Mur de briques	0.015	1.257
Trépieds	0.026	1.287
Boite de marbre	0.118	1.037
Chute d'une chaîne	−0.147	2.469

Ces relations linéaires peuvent servir pour apprendre le choix des valeurs des paramètres introduits dans l'un des algorithmes à point fixe en utilisant les techniques d'apprentissage automatiques. De plus, le tableau 3.2 montre que, dans la majorité des simulations, la moyenne des valeurs optimales de α pour chaque simulation est environ égale à 1.4. Ce résultat est en conformité avec les analyses réalisées par Erleben (2013). Dans ce cours, présenté à SIGGRAPH 2013, l'auteur a fait une déduction que la valeur moyenne dans une simulation donnée est égale à 1.4. Cette déduction était purement intuitive et basée sur des multiples essais de plusieurs valeurs du paramètre de relaxation. Cependant, ici, on a confirmé cette directive par une expérimentation plus rigoureuse.

Pour la simulation de la chute d'une chaîne métallique, la valeur moyenne est plus grande. Cela peut être expliqué par le fait que dans cette simulation on a des contraintes bilatérales inexistantes dans les autres simulations.

3.3.2.2 Nombre d'itérations

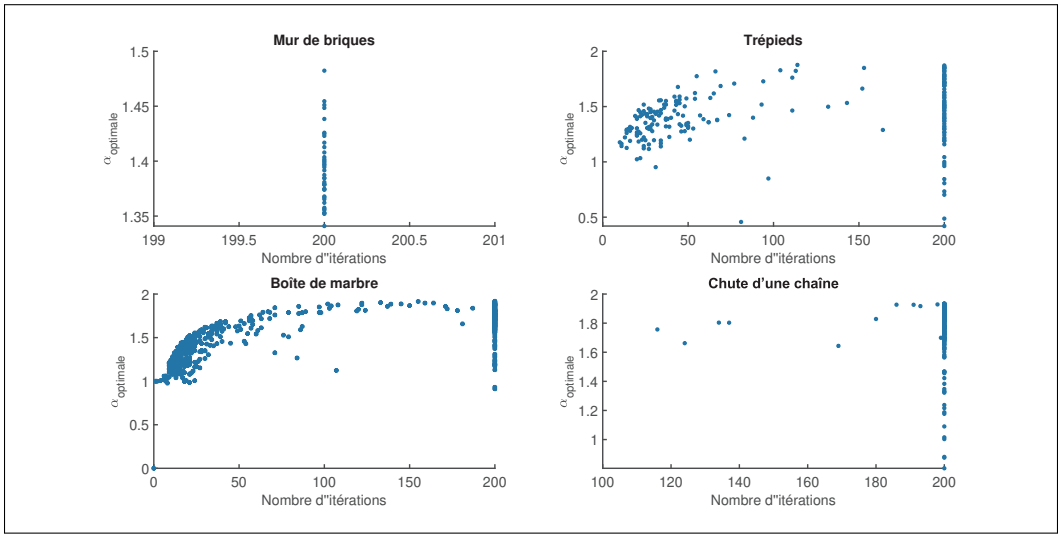


Figure 3.5 Valeurs de relaxation optimales en fonction de nombre d'itérations

En observant la figure 3.5, on remarque que lorsque le nombre d'itérations est inférieur au nombre d'itérations maximal, la valeur de relaxation optimale est supérieure à 1.0 dans la

majorité des systèmes résolus. Cela confirme l'idée de l'utilisation des valeurs de relaxation supérieure à 1.0 pour accélérer la convergence dans le cas d'un système convergent et des valeurs de relaxation inférieur à 1.0 pour ralentir la recherche vers une solution acceptable inférieure à la tolérance choisie. Dans la simulation de mur de briques, le nombre d'itérations maximale est toujours atteint. Cela est bien expliqué par le fait que les systèmes résolus dans cette simulation sont mal conditionnées par rapport aux autres simulations.

3.3.2.3 La taille du système

La figure 3.6 montre les représentations du paramètre de relaxation après optimisation en fonction les tailles de systèmes résolus. Les tailles des systèmes optimisées varient entre 21 et 2496.

Dans ses représentations, on n'a pas remarqué des relations apparentes entre le choix du meilleur paramètre et la taille de système résolu. On peut affirmer que le paramètre de relaxation n'est pas affecté par la taille de système dans la résolution avec PGS. Pour un exemple donné, notre choix de la meilleure valeur du paramètre de relaxation ne doit tenir compte de la taille de systèmes résolus.

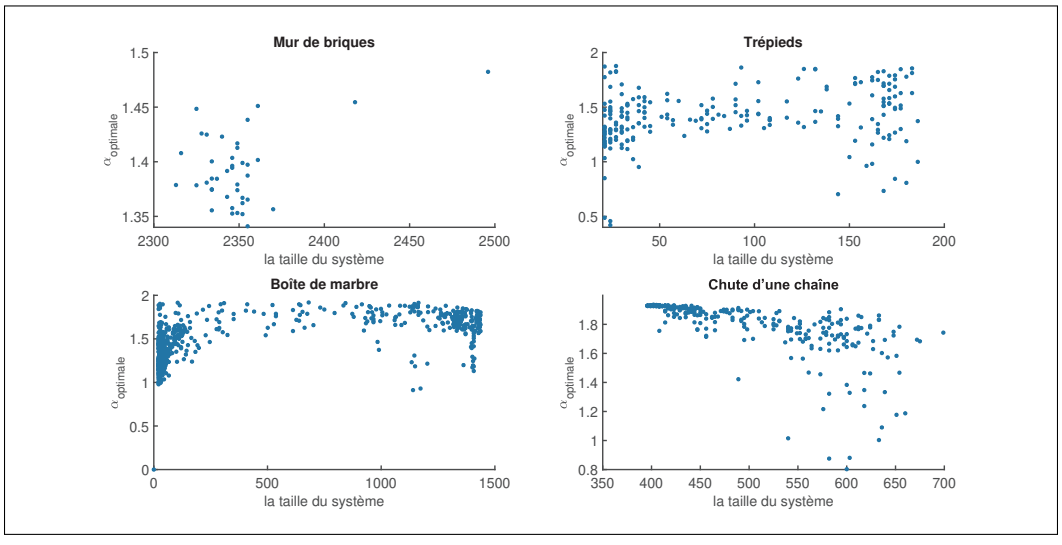


Figure 3.6 Valeurs de relaxation optimales en fonction de la taille du système

3.3.2.4 La norme du vecteur b

La figure 3.7 représente les valeurs optimales de α en fonction la norme de vecteur **b**. Cette représentation est supposée suivre l'évolution des valeurs de relaxation choisies en fonction de l'énergie cinétique dans le système. D'après les courbes dans cette figure, la variance dans les valeurs de relaxation choisies est plus importante dans le cas l'énergie du système est faible.

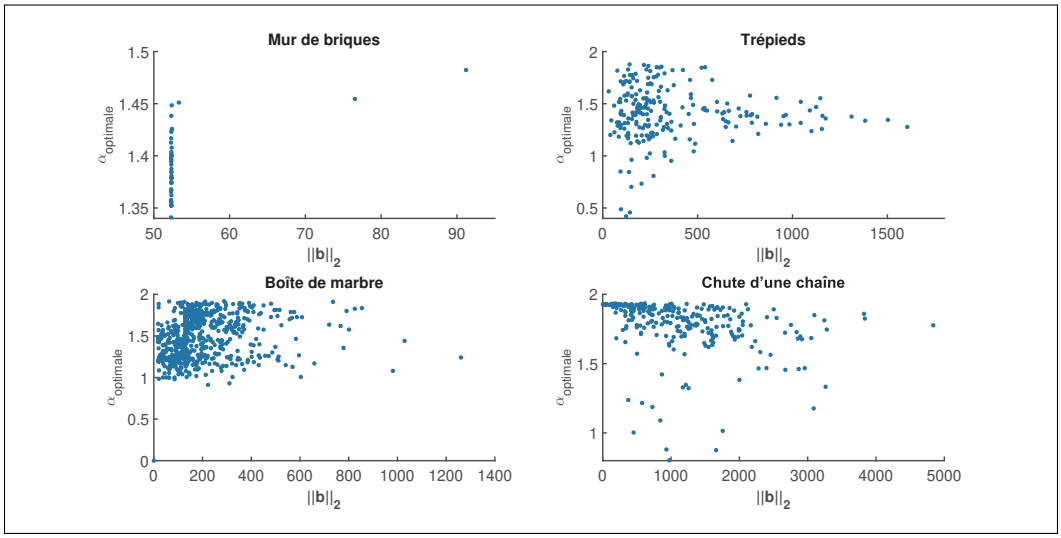


Figure 3.7 Valeurs de relaxation optimales en fonction de la norme du vecteur **b**

3.3.3 Optimisation du paramètre de régularisation

Dans cette section, on présente les résultats d'optimisation du paramètre de régularisation κ. En regardant les figures 3.10, 3.11, 3.9 et 3.8, on n'a pas trouvé une relation entre ces variables et les valeurs optimales choisies pour la régularisation.

Dans la figure 3.9, on remarque que, dans la plupart des systèmes résolus, la régularisation est ajoutée dans les systèmes qui n'atteint pas le nombre d'itérations maximal. Cela confirme l'effet de la régularisation dans l'amélioration de la convergence dans la résolution.

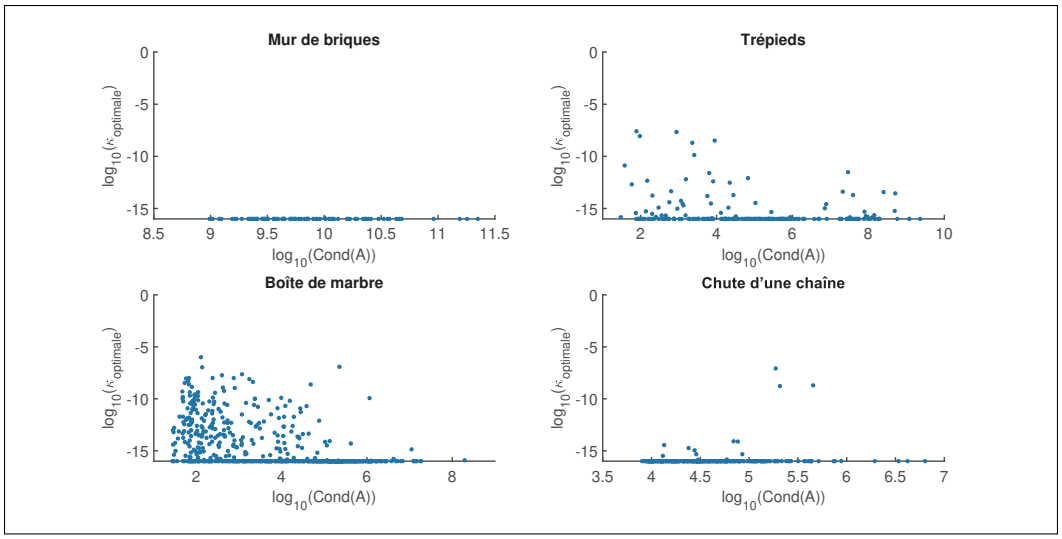


Figure 3.8 Valeurs de régularisation optimales en fonction du $\log_{10}$ du conditionnement de **A**

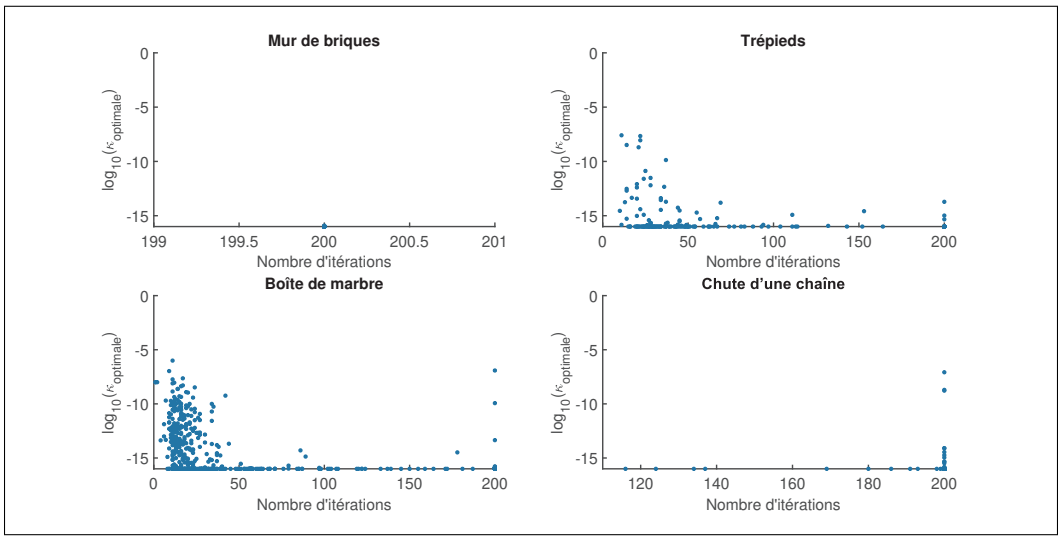


Figure 3.9 Valeurs de régularisation optimales en fonction de nombre d'itérations

La figure 3.10, montre l'évolution des valeurs optimales choisies pour le paramètre de régularisation pour les quatre simulations. On remarque que pour la plupart des systèmes dont les tailles inférieures à 1000, la régularisation est non-nulle.

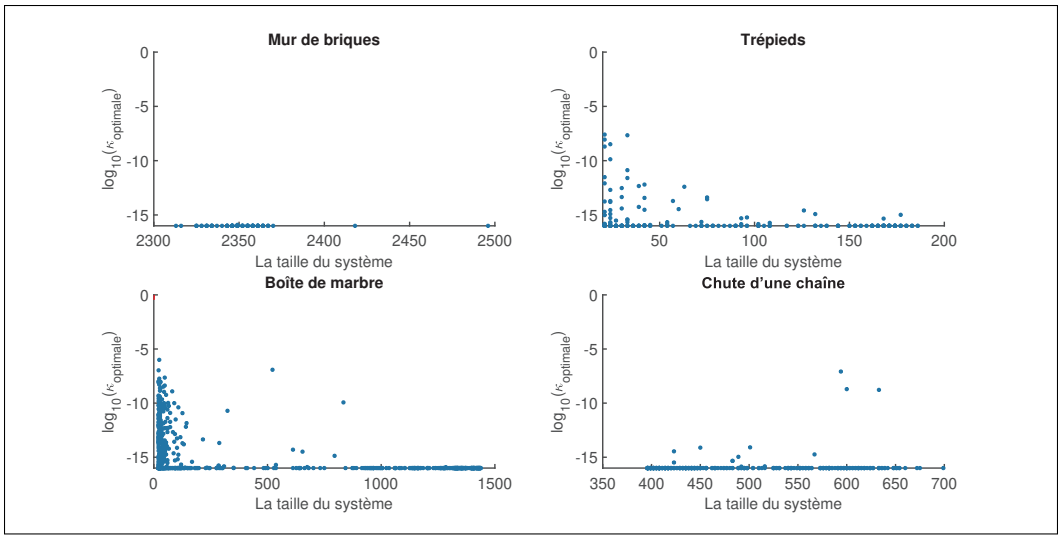


Figure 3.10 Valeurs de régularisation optimales en fonction de la taille du système

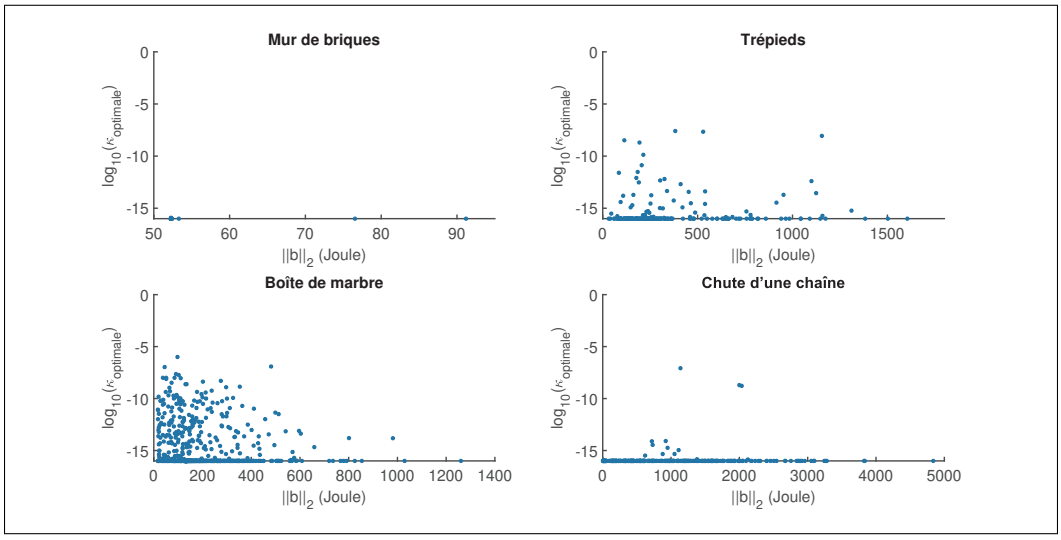


Figure 3.11 Valeurs de régularisation optimales en fonction de La norme du vecteur **b**

En observant la figure 3.11, on remarque les variations dans le choix des valeurs de régularisation sont plus importantes quand l'énergie cinétique du système est faible. On trouve un niveau d'énergie faible par rapport à une simulation au début et la fin de la simulation. Il est donc préférable d'ajouter de la régularisation ou de la relaxation dans notre résolution dans les images

de début et de la fin d'une simulation et on laisse les valeurs par défaut pour les images entre les deux.

3.4 Discussions

Les courbes dans la figure 3.12 représentant l'erreur au cours des différentes simulations montrent différentes améliorations d'une simulation à une autre. Dans la plupart des cas des systèmes optimisés, on voit une réduction de l'erreur par rapport à l'erreur générée par la résolution par défaut ($\alpha = 1.0$ et $\kappa = 0.0$), même si on n'atteint pas l'ordre d'erreur cherché au début. La réduction des erreurs permet de gagner en termes de précision du solveur, ce qui permet d'obtenir des simulations physiques précises et donc plus réalistes.

Le tableau 3.5 résume les réductions d'erreur générée dans les différentes simulations.

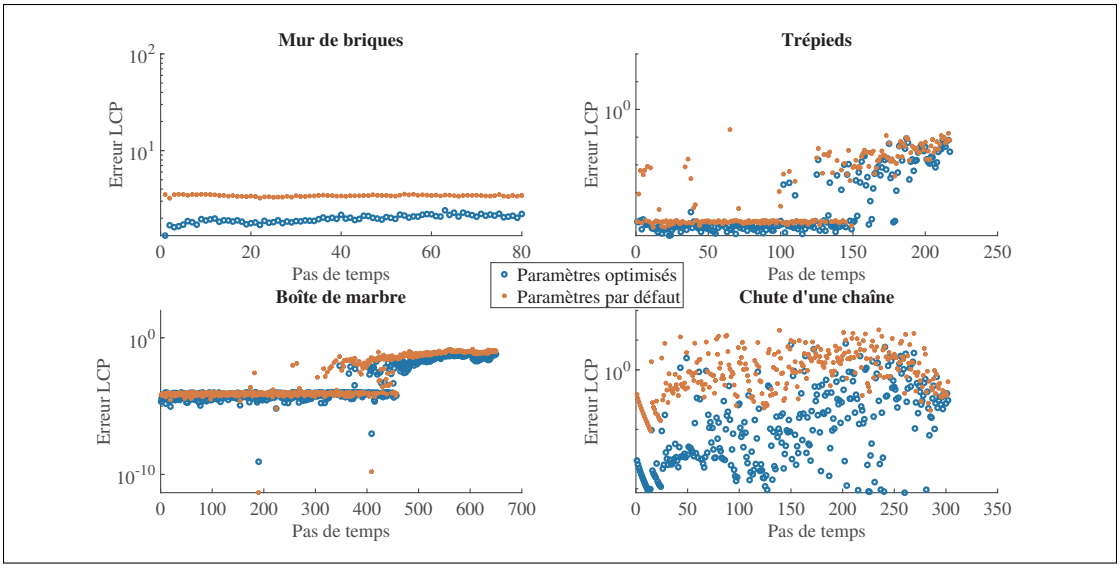


Figure 3.12 Erreur LCP avant et après l'optimisation

De plus, dans la plupart des cas, on observe une réduction dans le nombre totale d'itérations et l'erreur dans les simulations. Le tableau 3.6 indique que l'optimisation est bien fonctionnelle avec des pourcentages de réduction jusqu'à 22.07% dans le nombre total d'itérations dans une simulation. Seul, dans le cas de la simulation du mur, on n'a pas une réduction dans le

Tableau 3.5 Réduction d'erreur LCP dans les simulations avant (première ligne) et après (deuxième ligne) optimisation.

Simulations	**Min**	**Max**	**Moyenne**
Mur de briques	3.22e+00	3.56e+00	3.41e+00
	1.32e+00	3.84e+00	2.07e+00
Trépieds	5.67e-05	1.90e-01	1.47e-02
	2.83e-05	9.00e-02	7.20e-03
Boite de marbre	4.67e-12	1.48e-01	2.65e-02
	8.59e-10	8.14e-02	1.09e-02
Chute d'une chaîne	9.00e-03	22.14e+00	2.51e+00
	7.37e-05	7.77e+00	3.08e-01

nombre d'itérations, mais on remarque qu'il y a une amélioration dans l'ordre d'erreur. Cette réduction d'erreur démontre l'effet de l'optimisation appliquée sur les paramètres introduis dans l'algorithme PGS.

Tableau 3.6 Réduction du nombre d'itérations avec les paramètres optimaux

Simulations	**Pourcentage de réduction (%)**
Mur de briques	0.0
Trépieds	22.07
Boite de marbre	18.60
Chute d'une chaîne	0.62

L'effet de l'introduction de paramètres dans l'algorithme PGS et le bon choix des valeurs de ses paramètres pour chaque système résolu est bien justifié par la réduction du nombre d'itérations dans la plupart des simulations, mais surtout par la réduction de l'erreur de résolution afin d'atteindre la tolérance d'erreur cherchée (voir fig.3.12).

CONCLUSION ET RECOMMANDATIONS

La simulation interactive des corps rigides dans les moteurs physiques est souvent affectée par le type de solveur utilisé pour résoudre les contraintes cinématiques introduites. PGS est l'un des solveurs le plus populaire dans ce domaine et avoir une bonne convergence est toujours apprécié.

Dans ce mémoire, on cherche à améliorer le taux de convergence d'un tel type d'algorithme en introduisant des paramètres de régularisation et de relaxation. On choisit les bonnes valeurs pour avoir le meilleur réglage possible. Dans cette perspective, on a implémenté un processus automatique pour le réglage des paramètres introduits dans un solveur itératif, tel que PGS, afin d'avoir une meilleure convergence dans les limites fixées de tolérance d'erreur et de nombre d'itérations maximale. Afin de se rendre compte à cet objectif, on a commencé par implémenter la version classique de PGS puis on a introduit les paramètres dans les solveurs. Pour commencer l'optimisation des paramètres introduis, on a choisi l'algorithme d'optimisation CMA-ES. L'utilisation de cet algorithme d'optimisation permet de réduire l'erreur ainsi que les nombre d'itérations totales dans la plupart des simulations utilisées. Au cours du travail d'optimisation, on a rencontré une lenteur dans l'exécution du processus, c'est pourquoi on a remplacé la version classique du PGS par une version fondée sur les matrices creuses. Enfin, dans le but de réaliser nos analyses, on a créé un ensemble des courbes montrant les relations entre les paramètres introduits et quelques variables du système afin détecter les relations possibles entre eux.

Les expérimentations rigoureuses réalisées ont confirmé un résultat trouvé auparavant, la valeur de $\kappa = 1.4$ comme valeur pour le paramètre de relaxation est la bonne valeur dans la plupart des simulations, dans le cours de Erleben (2013) est correct sauf qu'il faut ajuster cette valeur en fonction du conditionnement du système résolu à chaque image de la simulation. Les relations entre le conditionnement du système et les valeurs choisis pour le paramètre de relaxation peuvent être approximées par des relations linéaires qui permettent de prédire les valeurs de paramètres pour les futures simulations. Aussi, remarque que l'introduction des contraintes

bilatérales dans une simulation rend la pente de la relation linéaire négative et que la valeur moyenne du paramètre de relaxation optimale augmente par rapport aux autres simulations. De plus, l'énergie cinétique dans la simulation peut affecter le choix des valeurs de ce deux paramètres introduis ; le changement des valeurs est plus remarquable dans les régions de faible énergie. Cela guide vers l'idée de changer les valeurs des paramètres introduits au début et à la fin d'une simulation puisqu'en général, pendant ces moments de la simulation, on trouve les niveaux les plus faibles d'énergie cinétique. Le choix de valeurs optimales pour les paramètres de relaxation et de régularisation n'est pas affectée par la taille du système résolue.

Comme travaux dans le futur, on peut continuer dans le même esprit en essayant le même pipeline en ajoutant d'autres paramètres tel que l'ordre de résolution, l'utilisation de l'option d'introduire la solution précédente dans la résolution (warm-starting), etc. Une autre idée consiste à créer un tableau de relations linéaires pour la valeur optimale des paramètres de relaxation et les conditionnements dans les systèmes résolus. Cela pourrait aider plus tard à créer un outil automatique pour choisir les meilleures valeurs pour le paramètre introduit en utilisant des algorithmes d'apprentissage automatique. Aussi, on peut étendre le modèle en prenant plus de variables telles que le type de contraintes, leur distribution, la proportion de contraintes actives. Cela permettra d'avoir un ensemble de données plus diversifié en plus de la possibilité d'avoir plus de corrélations entre les données.

D'autres applications de ce cadre d'optimisation peut être applique sur d'autres types de problèmes dans l'animation de corps rigides. Le travail de Liu & Andrews (2022) améliore les performances des méthodes de partitionnement pour les solveurs fondés sur la sous-structuration. L'amélioration peut être poussée plus loin en déterminant le nombre optimal des partitions par le même pipeline proposé dans ce mémoire. Il en va de même pour l'optimisation des hyper-paramètres pour le calcul des préconditionneurs dans certains types de solveurs numériques.

ANNEXE I

DESCRIPTION DU CMA-ES

L'algorithme CMA-ES est un algorithme aléatoire utilisé pour l'optimisation des paramètres réels dans le domaine continu de fonctions non linéaires et non convexes. Le processus d'optimisation est composée principalement de quatre étapes : échantillonnage, sélection et recombinaison, contrôle de la taille du pas et adaptation de la matrice de covariance.

1. Échantillonnage

Pour l'algorithme CMA-ES (Hansen, 2016), la génération d'une population de nouveaux points de recherche est réalisée en échantillonnant une distribution normale multivariable. L'équation qui définit l'échantillonnage pour plusieurs générations est :

$$\boldsymbol{x}_k^{(g+1)} \sim \boldsymbol{m}^{(g)} + \sigma^{(g)} \mathcal{N}\left(\mathbf{0}, \boldsymbol{C}^{(g)}\right) \quad \text{pour } k = 1, \ldots, \upsilon \text{ pour } g = 1, 2, \ldots \tag{A I-1}$$

Ou,

- $\mathcal{N}\left(\mathbf{0}, \boldsymbol{C}^{(g)}\right)$ est une distribution normale multivariable avec une moyenne nulle et une matrice de covariance $\boldsymbol{C}^{(g)}$.
- $\boldsymbol{x}_k^{(g+1)} \in \mathbb{R}^l$ est le k-ième descendant (individu, point de recherche) de la génération $g + 1$ avec l est le nombre de paramètres à optimiser.
- $\boldsymbol{m}^{(g)}$ est la moyenne de la distribution de recherche à la génération g.
- $\sigma^{(g)}$ est la taille du pas à la génération g.
- $\boldsymbol{C}^{(g)}$ est la matrice de covariance à la génération g.
- $\upsilon \geq 2$ est la taille de la population.

Les étapes qui suivent expliquent comment mettre à jour les quantités $\boldsymbol{m}^{(g+1)}$, $\boldsymbol{C}^{(g+1)}$ et $\sigma^{(g+1)}$ dans la prochaine génération $g + 1$.

2. Sélection et recombinaison

Après la sélection de meilleurs ξ points à partir υ points de la population générée dans la génération g, on doit changer la moyenne de la distribution de recherche. ξ est la taille de population des parents qui doit être inférieur a υ.

Pour la génération $g + 1$, la sélection de ξ points se fait en comparant la sortie de la fonction de fitness pour les υ points de la génération g :

$$\mathrm{f}(\boldsymbol{x}_1^{(g+1)}) \leq \mathrm{f}(\boldsymbol{x}_2^{(g+1)}) \leq \cdots \leq \mathrm{f}(\boldsymbol{x}_\upsilon^{(g+1)}).$$

Après la sélection de ξ points, on met à jour la valeur de la moyenne de la distribution des points à évaluer dans la génération $g + 1$ en utilisant l'équation (A I-2).

$$\boldsymbol{m}^{(g+1)} = \boldsymbol{m}^{(g)} + c_m \sum_{i=1}^{\xi} y_i \left(\boldsymbol{x}_{i:\upsilon}^{(g+1)} - \boldsymbol{m}^{(g)} \right). \tag{A I-2}$$

Avec,

- $c_m \leq 1$ est le taux d'apprentissage qui est en général égale à 1.
- $y_{i=1\ldots\xi} \in \mathbb{R}_{>0}$ sont des coefficients de poids positifs pour la recombinaison des points sélectionnés de la génération $g + 1$. Ces coefficients doivent vérifier $\sum_{i=1}^{\xi} y_i = 1$.
- $\boldsymbol{x}_{i:\upsilon}^{(g+1)}$ est le i-ème point sélectionné parmi les ξ points sélectionnés à partir des υ points de la population dans la génération $g + 1$.

3. Contrôle de la taille du pas

La taille du pas $\sigma^{(g)}$ est mis à jour à l'aide de l'adaptation de la taille des étapes (CSA). Tout d'abord, on met à jour *le chemin d'évolution conjugué* $\boldsymbol{p}_\sigma$ en appliquant l'équation (A I-3).

$$\boldsymbol{p}_\sigma^{(g+1)} = \overbrace{(1 - c_\sigma)}^{\text{facteur de réduction}} \boldsymbol{p}_\sigma^{(g)} + \overbrace{\sqrt{c_\sigma (2 - c_\sigma) \eta_{eff}}}^{\text{Constante de normalisation}} \boldsymbol{C}^{(g)^{-\frac{1}{2}}} \overbrace{\frac{\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}}}^{\text{déplacement de } \boldsymbol{m}} \tag{A I-3}$$

Avec,

- $\boldsymbol{p}_{\sigma}^{(g)} \in \mathbb{R}^{I}$ est le vecteur du chemin d'évolution conjugué à la génération g.
- $c_{\sigma} < 1$ avec c_{σ}^{-1} est l'horizon temporel rétrograde pour le chemin de l'évolution conjugué $\boldsymbol{p}_{\sigma}^{(g)}$.
- η_{eff} est la masse de sélection effective qui s'écrit comme suit $\eta_{eff} = \frac{1}{\sum_{i=1}^{\xi} y_i^2}$.
- $\boldsymbol{C}^{(g)^{-\frac{1}{2}}}$ est la racine carrée symétrique unique de l'inverse de $\boldsymbol{C}^{(g)}$.

Puis, on actualise la valeur de la taille du pas $\sigma^{(g+1)}$ à l'aide de l'équation (A I-4).

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_{\sigma}}{d_{\sigma}}\left(\frac{\left\|\boldsymbol{p}_{\sigma}^{(g+1)}\right\|}{\mathrm{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1\right)\right), \tag{A I-4}$$

où,

- $\mathrm{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|$ est l'espérance de la norme euclidienne d'un vecteur aléatoire qui suit la distribution $\mathcal{N}(\mathbf{0},\mathbf{I})$.
- d_{σ} est un facteur d'amortissement (la valeur par défaut est égale à 1).

4. Adaptation de la matrice de covariance

La dernière étape pendant une génération est d'adapter la matrice de covariance pour réduire la zone de recherche. Cette dernière étape commence par une mise à jour du de la valeur du *chemin d'évolution* $\boldsymbol{p}_{c}^{(g)}$ en utilisant l'équation (A I-5).

$$\boldsymbol{p}_{c}^{(g+1)} = \overbrace{(1-c_c)}^{\text{facteur de réduction}}\ \boldsymbol{p}_{c}^{(g)} + \overbrace{\sqrt{c_c\,(2-c_c)\,\eta_{eff}}}^{\text{Constante de normalisation}}\ \overbrace{\frac{\boldsymbol{m}^{(g+1)} - \boldsymbol{m}^{(g)}}{\sigma^{(g)}}}^{\text{déplacement de } \boldsymbol{m}} \tag{A I-5}$$

Avec,

- $\boldsymbol{p}_{c}^{(g)} \in \mathbb{R}^{I}$ est le vecteur du chemin d'évolution à la génération g.
- $c_c < 1$ avec c_c^{-1} est l'horizon temporel rétrograde pour le chemin de l'évolution $\boldsymbol{p}_{c}^{(g)}$.
- η_{eff} est la masse de sélection effective qui s'écrit comme suit $\eta_{eff} = \frac{1}{\sum_{i=1}^{\xi} y_i^2}$.

À la fin d'une génération, la mise à jour de la matrice de covariance se fait à l'aide de l'équation (A I-6).

$$\boldsymbol{C}^{(g+1)} = (1 \underbrace{-c_1 - c_\eta \sum y_j}_{\text{généralement égal à 0}})\boldsymbol{C}^{(g)} + c_1 \underbrace{\boldsymbol{p}_\mathrm{c}^{(g+1)} \boldsymbol{p}_\mathrm{c}^{(g+1)^\top}}_{\text{rank-one update}} + c_\eta \underbrace{\sum_{i=1}^{\upsilon} y_i \boldsymbol{h}_{i:\upsilon}^{(g+1)} \left(\boldsymbol{h}_{i:\upsilon}^{(g+1)}\right)^\top}_{\text{rank-}\eta\text{ update}}, \qquad \text{(A I-6)}$$

où,

- $c_1 \approx \frac{2}{l^2}$ avec l est le nombre de paramètres à optimiser.
- $c_\eta \approx min(\frac{\eta_{eff}}{l^2}, 1 - c_1)$
- $\boldsymbol{h}_{i:\upsilon}^{(g+1)} = \frac{(\boldsymbol{x}_{i:\upsilon}^{(g+1)} - \boldsymbol{m}^{(g)})}{\sigma^{(g)}}$

Algorithme-A I-1 L'algorithme CMA-ES

Définir des paramètres;

On va définir les paramètres initiaux υ,$y_{i=1\ldots\upsilon}$, c_σ,c_c,c_1 et c_η;

Initialisation;

On initialise les valeurs de $p_\sigma = 0$, $p_c = 0$ la matrice de covariance $\mathbf{C} = \mathbf{I}$ et $g = 0$;

On Choisit la moyenne et la variance (la taille du pas) de la distribution normale utilisée $\mathbf{m}$ et $\boldsymbol{\sigma}$;

while *critères d'arrêt non respectés* **do**

  Echantillonnage d'une nouvelle population de points de recherche;

  for $i = 1 \ldots \upsilon$ **do**

$$\mathbf{h}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{C})$$

$$\mathbf{x}_i \leftarrow \mathbf{m} + \sigma \mathbf{h}_i \sim \mathcal{N}\left(\mathbf{m}, \sigma^2 \boldsymbol{C}\right)$$

  end for

  Sélection et recombinaison;

$$\bar{\mathbf{h}} \leftarrow \sum_{i=1}^{\xi} y_i \mathbf{h}_{i:\upsilon}$$

$$\mathbf{m} \leftarrow \mathbf{m} + c_m \sigma \bar{\mathbf{h}}$$

  Contrôle de la taille du pas;

$$\boldsymbol{p}_\sigma \leftarrow (1 - c_\sigma)\, \boldsymbol{p}_\sigma + \sqrt{c_\sigma\,(2 - c_\sigma)\,\eta_{eff}}\; \boldsymbol{C}\frac{\bar{\boldsymbol{h}}}{\sigma}$$

$$\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma\|}{\mathrm{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1\right)\right)$$

  Adaptation de la matrice de covariance;

$$\boldsymbol{p}_c \leftarrow (1 - c_c)\, \boldsymbol{p}_c + \sqrt{c_c\,(2 - c_c)\,\eta_{eff}}\; \frac{\bar{\boldsymbol{h}}}{\sigma}$$

$$\boldsymbol{C} = (1 - c_1 - c_\eta \sum y_j)\boldsymbol{C} + c_1 \boldsymbol{p}_{\mathrm{c}} \boldsymbol{p}_{\mathrm{c}}^{\top} + c_\eta \sum_{i=1}^{\upsilon} y_i \boldsymbol{h}_{i:\upsilon} \left(\boldsymbol{h}_{i:\upsilon}\right)^{\top}$$

  $g \leftarrow g + 1$;

end while

ANNEXE II

LISTE DES PARAMÈTRES DU CMA-ES

Tableau-A II-1 Table des paramètres du CMA-ES

Nom du paramètre	Description	Valeurs
LBounds	Limites inférieures du vecteur des paramètres	[0.1 ; -16]
UBounds	Limites supérieures du vecteur des paramètres	[2.0 ; -1]
MaxFunEvals	Nombre maximal d'évaluations de fonctions	Inf
PopSize	La taille de la population initiale	24
EvalParallel	Option qui permet les évaluations en parallèle	'yes'
EvalInitialX	Option pour évaluer la solution initiale	'yes'
SaveVariables	Option pour sauvegarder les variables à la fin de l'optimisation	'on'
ParentNumber	La nombre de candidats à choisir pour mettre à jour la moyenne et la variance de la distribution des paramètres	12
StopFitness	La valeur minimale de la fonction de fitness	10^{-9}
TolX	On s'arrête si le changement dans les paramètres est plus petite que cette valeur	$10^{-11}\max(\boldsymbol{\sigma})$
TolUpX	On s'arrête si le changement dans les paramètres est plus grand que cette valeur	$10^{3}\max(\boldsymbol{\sigma})$
TolFun	La tolérance sur les valeurs de la fonction de fitness	10^{-12}
TolHistFun	La tolérance sur l'historique des changements des valeurs de la fonction de fitness	10^{-13}
StopOnStagnation	Option pour s'arrêter lorsque la fonction de fitness stagne pendant une longue période	'on'
Restarts	Nombre de redémarrage pour la recherche des solutions	0

ANNEXE III

LES VISUALISATIONS DU PROCESSUS D'OPTIMISATION AVEC CMA-ES

La convergence de CMA-ES a été contrôlée à l'aide de plusieurs figures.

- **Subplot gauche en haut :**
  - Bleu : valeur de la fonction de la meilleure solution dans la population récente.
  - Cyan : même valeur de fonction moins la meilleure valeur de fonction jamais vue.
  - Vert : sigma (la variance de la fonction de fitness entre une itération et une autre)
  - Rouge : rapport entre la longueur de l'axe principal le plus long et le plus court (ce qui donne une idée de la taille de la zone de recherche).
- **Subplot droite en haut :** La figure décrit l'évolution temporelle de la moyenne de la distribution (par défaut) ou du vecteur de la meilleure solution récente.
- **Subplot gauche en bas :** La figure présente les longueurs des axes principaux de l'ellipsoïde de distribution, équivalentes aux valeurs propres de racine carrée de $\mathbf{C}$.
- **Subplot droite en bas :** La figure présente l'écart-type "vrai" minimal et maximal (avec sigma inclus) dans les coordonnées, autres couleurs : $\sqrt{\text{diag}(\mathbf{C})}$ de tous les éléments diagonaux de $\mathbf{C}$, si $\mathbf{C}$ est diagonal, ils sont égaux en bas à gauche.

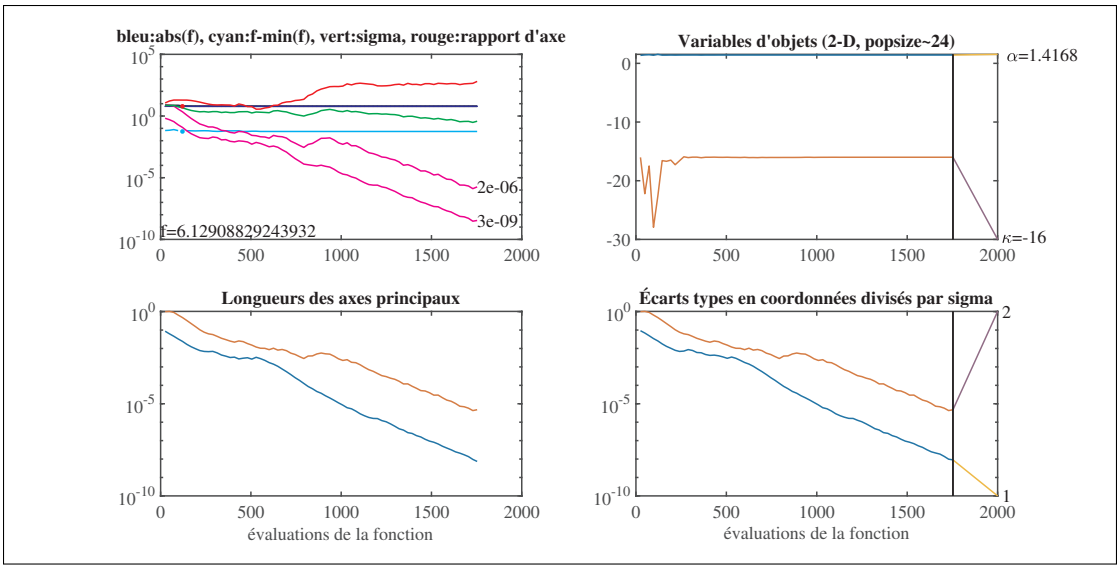


Figure-A III-1 Évaluation de la fonction de fitness pour la simulation de mur de briques

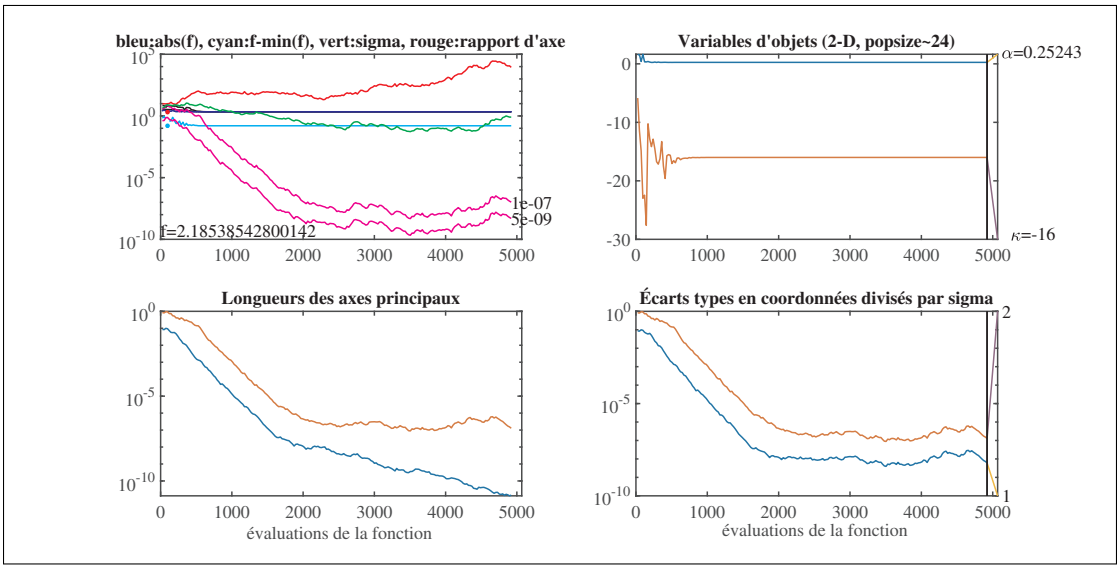


Figure-A III-2 Évaluation de la fonction de fitness pour la simulation de trépieds

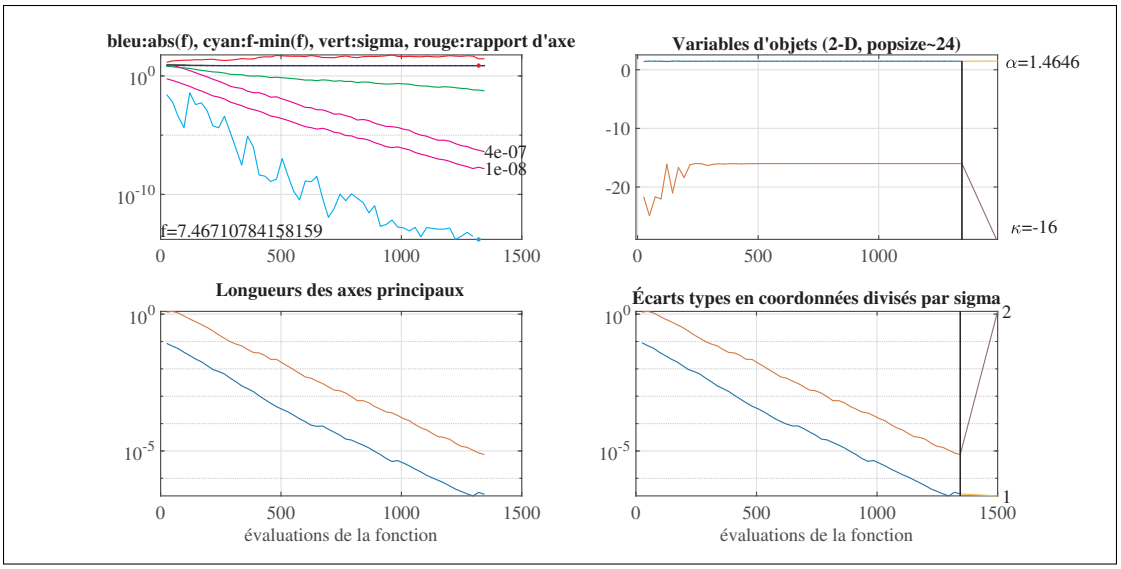


Figure-A III-3 Évaluation de la fonction de fitness pour la simulation de boîte de marbre

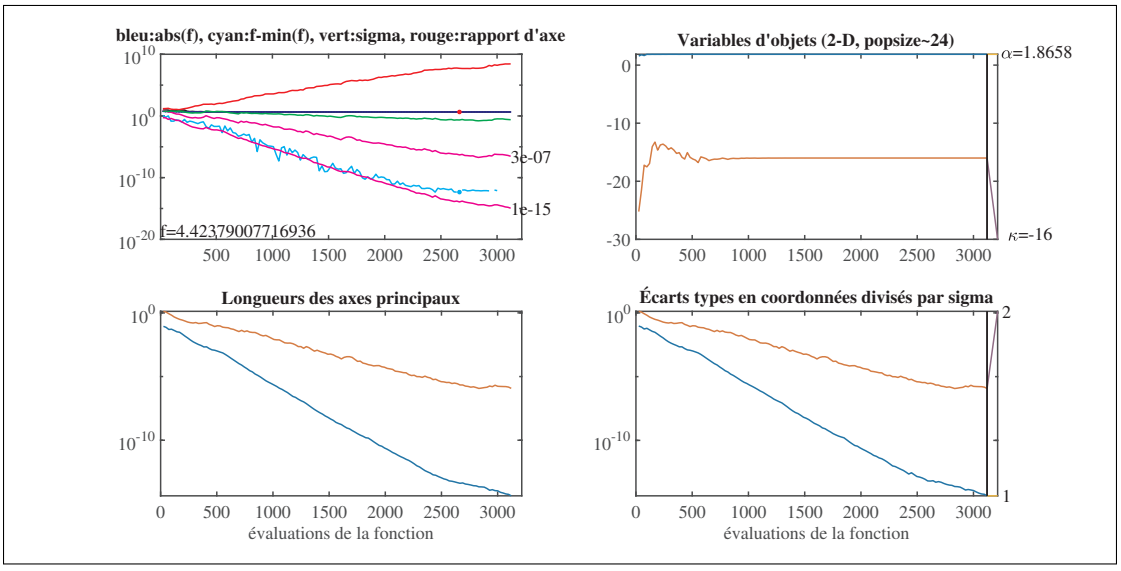


Figure-A III-4 Évaluation de la fonction de fitness pour la simulation de chute d'une chaîne

BIBLIOGRAPHIE

Andrews, S. & Erleben, K. (2021). Contact and friction simulation for computer graphics. *ACM SIGGRAPH 2021 Courses*, (SIGGRAPH '21), 1–124. doi : 10.1145/3450508.3464571.

Andrews, S., Erleben, K., Kry, P. G. & Teichmann, M. (2017). Constraint reordering for iterative multi-body simulation with contact. *ECCOMAS Thematic Conference on Multibody Dynamic*, pp. 18–22.

Baumgarte, J. (1972). Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1), 1–16. doi : 10.1016/0045-7825(72)90018-7.

Bender, J., Erleben, K. & Trinkle, J. (2014). Interactive Simulation of Rigid Body Dynamics in Computer Graphics. *Computer Graphics Forum*, 33(1), 246–270. doi : 10.1111/cgf.12272.

Buluç, A., Fineman, J. T., Frigo, M., Gilbert, J. R. & Leiserson, C. E. (2009). Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures - SPAA '09*, pp. 233. doi : 10.1145/1583991.1584053.

Cline, M. & Pai, D. (2003). Post-stabilization for rigid body simulation with contact and constraints. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 3, 3744–3751 vol.3. doi : 10.1109/ROBOT.2003.1242171.

CM Labs Simulations. (2019). Vortex Studio. Repéré à https://www.cm-labs.com/vortex-studio/.

Enzenhöfer, A., Andrews, S., Teichmann, M. & Kövecses, J. (2018). Comparison of Mixed Linear Complementarity Problem Solvers for Multibody Simulations with Contact. 10. doi : 10.2312/vriphys.20181063.

Enzenhöfer, A., Lefebvre, N. & Andrews, S. (2019). Efficient block pivoting for multibody simulations with contact. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 1–9. doi : 10.1145/3306131.3317019.

Erleben, K. (2004). Stable, robust, and versatile multibody dynamics animation. *Unpublished Ph. D. Thesis, University of Copenhagen, Copenhagen*, 1–241.

Erleben, K. (Éd.). (2005). *Physics-based animation* (éd. 1st ed). Hingham, Mass : Charles River Media. doi : 10.5555/1051390.

Erleben, K. (2007). Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics (TOG)*, 26(2), 12. doi : 10.1145/1243980.1243986.

Erleben, K. (2013). Numerical Methods for Linear Complementarity Problems in Physics-based Animation. 42. doi : 10.2200/S00621ED1V01Y201412CGR018.

Fratarcangeli, M., Tibaldo, V. & Pellacini, F. (2016). Vivace : a practical gauss-seidel method for stable soft body dynamics. *ACM Transactions on Graphics (TOG)*, 35(6), 1–9. doi : 10.1145/2980179.2982437.

Goldstein, H., Poole, C. P. & Safko, J. L. (2008). *Classical mechanics* (éd. 3. ed., [Nachdr.]). San Francisco Munich : Addison Wesley.

Hansen, N. (2016). The CMA Evolution Strategy : A Tutorial. *arXiv :1604.00772 [cs, stat]*, 1–39. Repéré à http://arxiv.org/abs/1604.00772.

Hansen, N., Müller, S. D. & Koumoutsakos, P. (2003). Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1), 1–18. doi : 10.1162/106365603321828970.

Hansen, N., Auger, A., Ros, R., Finck, S. & Pošík, P. (2010). Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, (GECCO '10), 1689–1696. doi : 10.1145/1830761.1830790.

Júdice, J. J. & Pires, F. M. (1994). A block principal pivoting algorithm for large-scale strictly monotone linear complementarity problems. *Computers & Operations Research*, 21(5), 587–596. doi : 10.1016/0305-0548(94)90106-6.

Koranne, S. (2010). Hierarchical Data Format 5 : HDF5. Dans *Handbook of Open Source Tools* (pp. 191–200). Springer US. doi : 10.1007/978-1-4419-7719-9_10.

Liu, Y. & Andrews, S. (2022). Graph Partitioning Algorithms for Rigid Body Simulations. *Eurographics 2022 (Short Papers)*.

Lloyd, J. (2005). Fast Implementation of Lemke's Algorithm for Rigid Body Contact Simulation. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 4538–4543. doi : 10.1109/ROBOT.2005.1570819.

Mazhar, H., Heyn, T., Negrut, D. & Tasora, A. (2015). Using Nesterov's Method to Accelerate Multibody Dynamics with Friction and Contact. *ACM Transactions on Graphics (TOG)*, 34(3), 32 :1–32 :14. doi : 10.1145/2735627.

Miyamoto, S. & Yamashita, M. (2021). An improved convergence based on accelerated modulus-based Gauss–Seidel method for interactive rigid body simulations. *SN Applied Sciences*, 3(2), 266. doi : 10.1007/s42452-021-04238-8.

Peiret, A., Andrews, S., Kövecses, J., Kry, P. G. & Teichmann, M. (2019). Schur Complement-based Substructuring of Stiff Multibody Systems with Contact. *ACM Transactions on Graphics (TOG)*, 38(5), 1–17. doi : 10.1145/3355621.

Poulsen, M., Niebe, S. & Erleben, K. (2010). Heuristic Convergence Rate Improvements of the Projected Gauss-Seidel Method for Frictional Contact Problems. *Proceedings of World Society for Computer Graphics (WSCG)*.

Silcowitz, M., Niebe, S. & Erleben, K. (2010). Projected Gauss-Seidel Subspace Minimization Method for Interactive Rigid Body Dynamics - Improving Animation Quality using a Projected Gauss-Seidel Subspace Minimization Method. 229, 38–45. doi : 10.1007/978-3-642-25382-9_15.

Silcowitz-Hansen, M., Niebe, S. & Erleben, K. (2010). A nonsmooth nonlinear conjugate gradient method for interactive contact force problems. doi : 10.1007/s00371-010-0502-6.

Wang, H. (2015). A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)*, 34(6), 1–9. doi : 10.1145/2816795.2818063.

Young, D. M. (1971). *Iterative Solution of Large Linear Systems*. Elsevier Inc, Academic Press. doi : 10.1016/c2013-0-11733-3.