Université de Liège

Stabilité d'un pont suspendu soumis à des sollicitations cycliques

Introduction aux méthodes numériques et projet

Bastien Hoffmann
Maxime Meurisse
François Rozet

1ère année de Bachelier Ingénieur civil
Année académique 2016 - 2017

Table des matières

1 Évolution de l'enveloppe de l'angle de torsion

Le problème qui nous est posé est de déterminer le plus précisément possible après combien de temps l'angle de torsion du tablier vaudra 0,3 radians. Pour cela, un fichier `enveloppe.xls` contenant des valeurs de l'angle à différents temps est mis à notre disposition. Ne connaissant pas la relation analytique liant ces deux paramètres, nous devons interpoler les valeurs données et déterminer grâce à la fonction obtenue le temps correspondant à l'angle recherché.

1.1 L'interpolation

Pour effectuer l'interpolation sur les valeurs données, notre choix s'est porté sur la méthode des *splines cubiques*. Celle-ci a pour particularité de réaliser une interpolation par morceaux, comblant l'intervalle entre chaque pair de points par une fonction du 3$^{\text{ème}}$ degré. Cela assure la continuité de la courbe et l'absence de points anguleux. De plus, avec ce procédé, même sur un grand nombre de points, la possibilité de voir apparaître des oscillations non justifiées est quasi nulle, contrairement à l'interpolation polynômiale.

Après implémentation de la méthode sur le logiciel MATLAB [1], nous obtenons le résultat suivant.

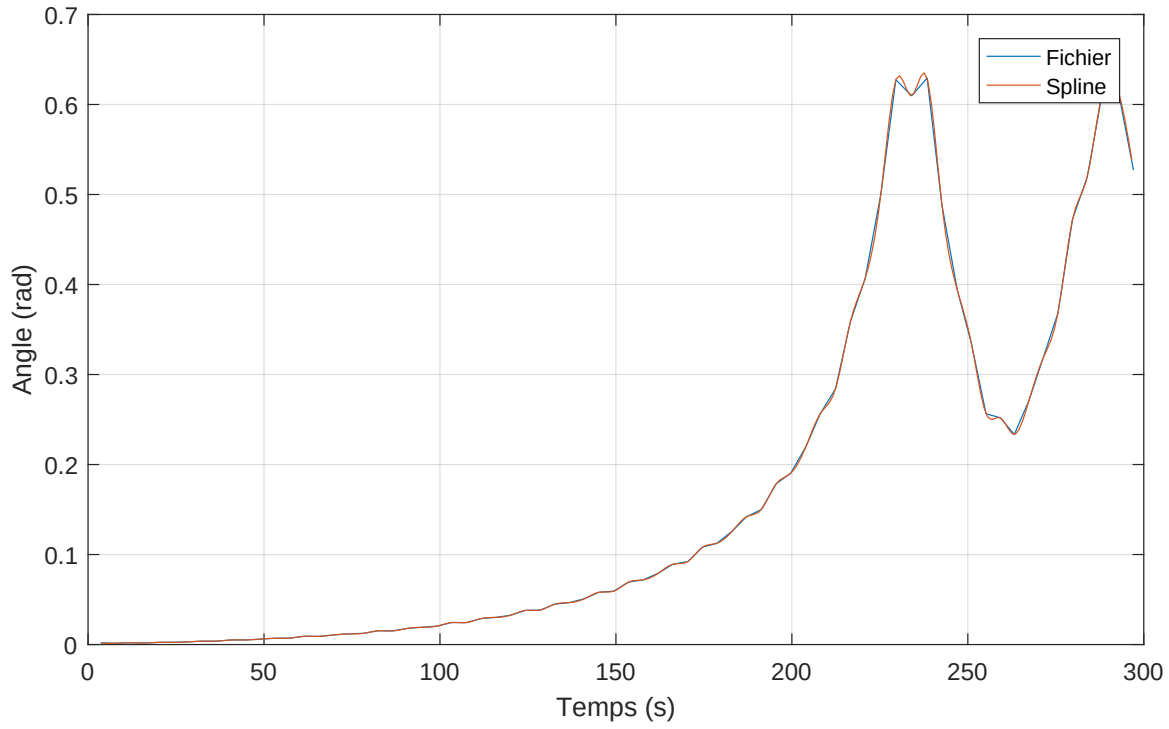

Figure 1 – Interpolation par *splines cubiques* des valeurs du fichier `enveloppe.xls`

1. Tous les algorithmes présentés dans ce rapport seront implémentés dans MATLAB R2016a et exécutés sur un ordinateur tournant sous Windows, doté d'un processeur Intel i7 cadencé à 4 GHz et possédant 32 GB de RAM.

Nous constatons que l'interpolation est concluante car elle se rapproche très fortement des valeurs du fichier `enveloppe.xls`.

1.2 Étude de la fonction obtenue

Afin de déterminer le plus précisément possible le temps recherché, nous avons implémenté deux méthodes qui permettent de rechercher, sous certaines tolérances, une abscisse correspondant à une image visée et cela à partir de deux abscisses/bornes de départ x_0 et x_1.

Pour choisir celles-ci, nous avons implémenté une fonction assez simple. En premier lieu, l'algorithme de la fonction réduit toutes les images du fichier, préalablement enregistrées dans une matrice, par l'image visée. Il considère, ensuite, le produit de l'image de chaque point avec celle du précédent. Si les deux points se trouvent du « même côté » de l'ordonnée visée, le produit est positif et inversement. Puisqu'il nous faut deux points dont les images sont de part et d'autres de celle visée, l'algorithme s'arrête aux premiers points pour lesquels le produit est négatif et il désigne leur abscisse comme bornes de départ.

Ce système à, la plupart du temps, comme avantage de coincer, entre les bornes de départ, uniquement la première abscisse correspondant à l'ordonnée demandée.

1.2.1 La méthode de la bissection

Cette méthode consiste en une *recherche dichotomique*. Elle travaille dans un intervalle qu'elle réduit de moitié à chaque itération. Cependant, elle veille toujours à garder l'abscisse recherchée dans l'intervalle, convergeant ainsi vers sa cible et permettant de fixer une tolérance sur celle-ci.

Pour utiliser cette démarche, nous devons vérifier deux hypothèses. Premièrement, la fonction est-elle continue ? En ce qui nous concerne, cette condition est bien respectée grâce à l'interpolation par splines cubiques. Secondement, les images des abscisses de départ sont-elles de part et d'autre de l'image visée ? Celle-ci aussi est respectée par l'utilisation de la fonction implémentée pour déterminer les bornes.

Puisque les hypothèses sont respectées, l'algorithme convergera assurément vers l'abscisse recherchée. Cependant, son taux de convergence étant linéaire, elle y parviendra assez lentement.

1.2.2 La méthode de la sécante

Cet autre procédé se base sur une relation de récurrence. À partir de deux abscisses x_n et x_{n+1}, une abscisse supplémentaire x_{n+2} est déterminée comme l'abscisse du point de la droite X_nX_{n+1} dont l'image est égale à celle visée. L'opération est ensuite répétée avec les points x_{n+1} et x_{n+2} et ainsi de suite jusqu'à s'être approché suffisamment de l'ordonnée visée.

Le bon fonctionnement de ce système ne peut pas être garanti par un critère simple. Cependant lorsque la fonction est suffisament monotone sur l'intervalle où la méthode travaille, il y a beaucoup de chance qu'elle trouve l'abscisse recherchée. De plus, la méthode de la sécante possède un ordre de convergence super-linéaire. Elle la trouvera donc plus vite que la méthode de la bissection dans la grande majorité des cas.

1.2.3 Critère d'arrêt et tolérances

Pour les deux démarches présentées ci-dessus, il est important de fixer un critère d'arrêt cohérent et des tolérances adéquates. Notre algorithme fonctionne de la sorte : tant que le critère d'arrêt n'est pas atteint, et que l'une des tolérances n'est pas respectée, la boucle continue.

Nous avons déterminé de manière empirique toutes nos tolérances, c.-à-d. que nous les avons toutes fixées à l'epsilon machine, puis nous les avons doublées séparément jusqu'à trouver les plus petites valeurs qui permettent à nos algorithmes de converger. Ainsi, nous avons, pour la sécante, une précision sur les ordonnées de `eps` et, pour la bissection, une précision sur les ordonnées de `4eps` et sur les abscisses de `32eps`.

Le critère d'arrêt de la bissection est calculé au départ de la fonction de telle manière que l'intervalle sur lequel elle travaille ne puisse pas être plus petit que `eps`. Quant à celui de la sécante il a été fixé arbitrairement à 32.

1.2.4 Conclusion

Après implémentation, nous avons obtenu les résultats suivants :

- la méthode de la bissection a donné un temps d'environ [2] 213 secondes après 46 itérations.
- la méthode de la sécante a donné le même temps après 8 itérations.

Nous constatons donc bien que les deux procédés convergent vers un même temps qui, par vérification graphique, semble bien être la solution recherchée. La méthode de la bissection étant plus robuste lorsque ces conditions initiales sont respectées, notre choix se portera sur celle-ci pour la suite du problème.

2 Simulation du système du pont suspendu

Cette deuxième question ayant pour but de simuler le système du pont, la modélisation de son comportement se base sur les équations linéaire et non-linéaire fournies. Afin de les résoudre, les deux solveurs employés sont la méthode d'Euler explicite et d'`ode45`. Le système d'équations différentielles du deuxième ordre qui est soumis à ces deux démarches se base sur les principales variables du mouvement du pont, à savoir sa position y et l'angle

2. 213,451 369 648 658 0 s

d'inclinaison θ de son tablier, celles-ci étant ré-exprimées comme variables des équations données.

2.1 Ré-écriture du système d'équations différentielles

Avant toute chose, nous avons commencé par ré-écrire le système d'équations différentielles donné. Par le changement de variables

$$\begin{cases} y_+ = y + l\sin\theta \\ y_- = y - l\sin\theta \end{cases}$$

nous obtenons, dans le cas du modèle linéaire

$$\begin{cases} y'' = \left(A\sin(\omega t) - 2Ky - c_y y'\right)\dfrac{1}{m} \\ \theta'' = \left(3Kl^2\sin(2\theta) + 3c_\theta\theta'\right)\dfrac{1}{-ml^2} \end{cases}$$

et dans le cas du modèle non linéaire

$$\begin{cases} y'' = \left(A\alpha\sin(\omega t) - K(e^{\alpha(y+l\sin\theta)} + e^{\alpha(y-l\sin\theta)} - 2) - \alpha c_y y'\right)\dfrac{1}{\alpha m} \\ \theta'' = \left(Kl\theta(e^{\alpha(y-l\sin\theta)} - e^{\alpha(y+l\sin\theta)}) - \alpha c_\theta\theta'\right)\dfrac{3}{ml^2\alpha} \end{cases}$$

2.2 Méthode d'Euler explicite

Cette méthode ne bénéficiant pas déjà d'une implémentation dans MATLAB, nous l'avons réalisée nous-même. Le principe de cette dernière se base sur un développement de Taylor à l'ordre 1.

Pour obtenir une erreur acceptable sur les valeurs obtenues, nous avons envisagé un pas de 10^{-3}. Déterminé de manière empirique, ce pas est suffisant pour obtenir un modèle stable. L'algorithme prenant un temps considérable pour calculer l'ensemble des valeurs, nous nous sommes contentés du premier pas fournissant des résultats stables afin de ne pas devoir laisser tourner la boucle trop longtemps.

2.3 Méthodes de *Runge-Kunta*

Les méthodes de *Runge-Kutta* offrent un large panel de choix en ce qui concerne leur utilisation dans un solveur de systèmes d'équations différentielles. Nous employons dans le cas présent la méthode d'`ode45`. A l'instar d'Euler explicite et de tous les autres solveurs `ode`, `ode45` utilise un développement de Taylor afin de résoudre les systèmes d'équations qui lui sont soumis.

Ces procédés se distinguent essentiellement les uns des autres par l'ordre du développement de Taylor des différents résultats évalués lors de la résolution. Dans le cas présent,

`ode45` est une méthode adaptative, considérant un développement de Taylor à l'ordre 4 et d'un second à l'ordre 5. Cette combinaison le rend plus efficace et plus précis. Contrairement à la méthode d'Euler explicite, le pas utilisé n'est pas constant et est déterminé par le solveur. Ce dernier l'adaptera en fonction des tolérances fixées. Celles de base utilisées par `ode45` sont de 10^{-3} pour `RelTol` et 10^{-6} pour `AbsTol`. Après vérification, il s'est avéré qu'augmenter la précision de `RelTol` à 10^{-4} n'apportait rien. Nous avons donc conservé les tolérances initiales.

2.4 Comparaison des solveurs

Effectuons la résolution demandée sur un intervalle de temps allant de 0 à 1000 secondes afin de comparer les deux procédés.

Tout d'abord, nous pouvons constater une différence d'efficacité marquée entre les deux démarches. Toutes les deux conduisent à des résultats assez similaires, mais lorsque nous considérons les temps d'exécutions (pour la position du tablier par exemple), nous observons de larges différences. Grâce à l'utilisation du profiler, il s'avère que la méthode d'Euler explicite prend beaucoup plus de temps que celle d'`ode45`.

Function name	**Calls**	**Total time**	**Self Time**
euler_exp	1	7,921 s	3,301 s
odefunction_nl	1000000	4,620 s	4,620 s
ode45	1	0,276 s	0,145 s
odefunction_nl	15913	0,093 s	0,093 s

Tableau 1 – Comparaison des temps d'exécution des méthodes Euler explicite et `ode45` pour le modèle non-linéaire (calcul de la position du tablier).

Ensuite, comparons les résultats obtenus. Les comportements déterminés par les différents solveurs sont peu plausibles et pratiquement similaires. En effet, en ce qui concerne la position du tablier, celle-ci présente des oscillations très importantes dans un premier temps avec un déplacement maximal de près de 6 mètres par rapport à sa position d'équilibre. Quelque soit le modèle observé, les oscillations tendent à se stabiliser avec un déplacement de 1,5 m autour de la position d'équilibre.

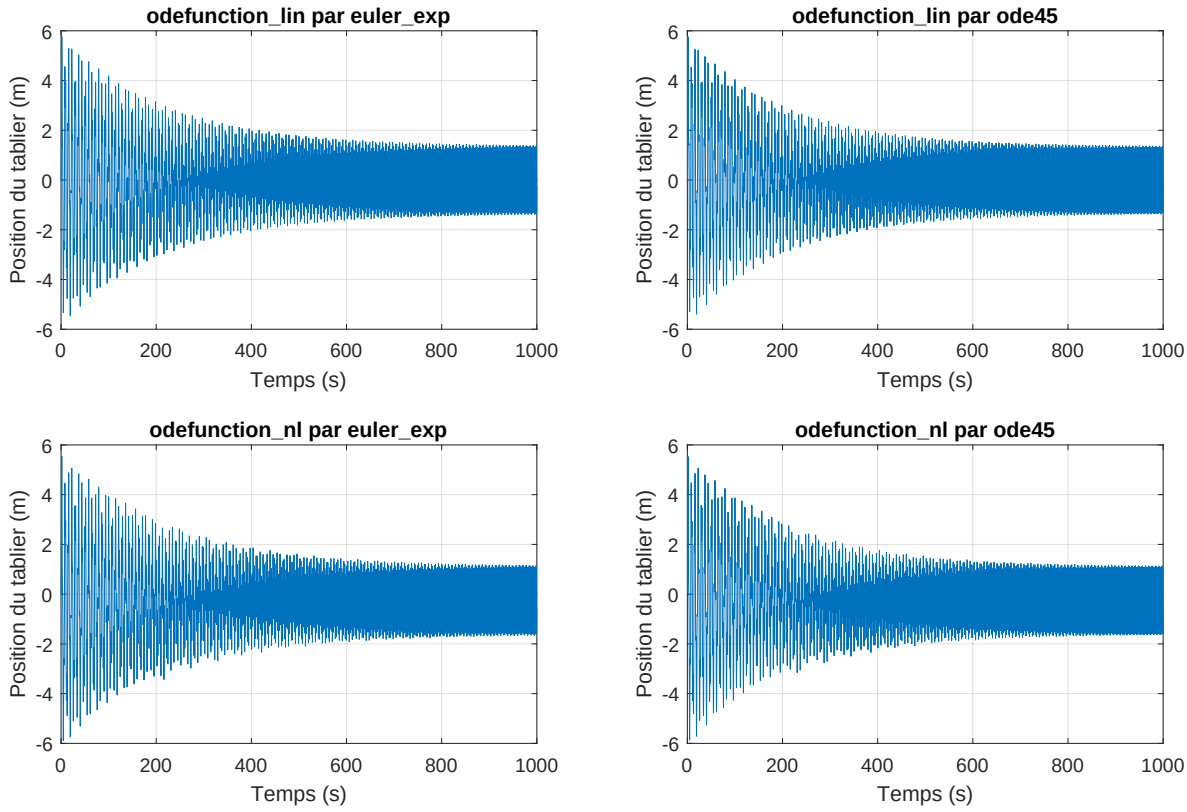

Figure 2 – Variation de la position du tablier pour les modèles linéaire et non-linéaire, résolus avec les méthodes Euler explicite et `ode45`

Observons également l'évolution de l'inclinaison du tablier. Dans cette situation, les différentes méthodes sont à nouveau pratiquement similaires, les comportements de l'inclinaison sont identiques dans les deux procédés. Cependant, on constate une divergence totale suivant le modèle utilisé. Dès le départ, le modèle linéaire décrit de faibles perturbations qui décroissent assez rapidement et se stabilisent à une valeur encore plus faible. Inversement, le modèle non-linéaire considère de faibles oscillations initiales du tablier avant que celles-ci ne se mettent à augmenter drastiquement. Au contraire des graphiques précédents, nous pouvons aisément constater ici les limitations du modèle linéaire. Ses simplifications conduisent à un résultat qui parait éloigné de la réalité pour un vent cyclique d'une telle force. Le modèle non-linéaire, plus précis dans ses évaluations de la situation, nous fournit un comportement beaucoup plus cohérent avec ce que pourrait subir le tablier comme mouvements, c.-à-d. une oscillation continue entre 0,5 rad et $-0{,}5$ rad.

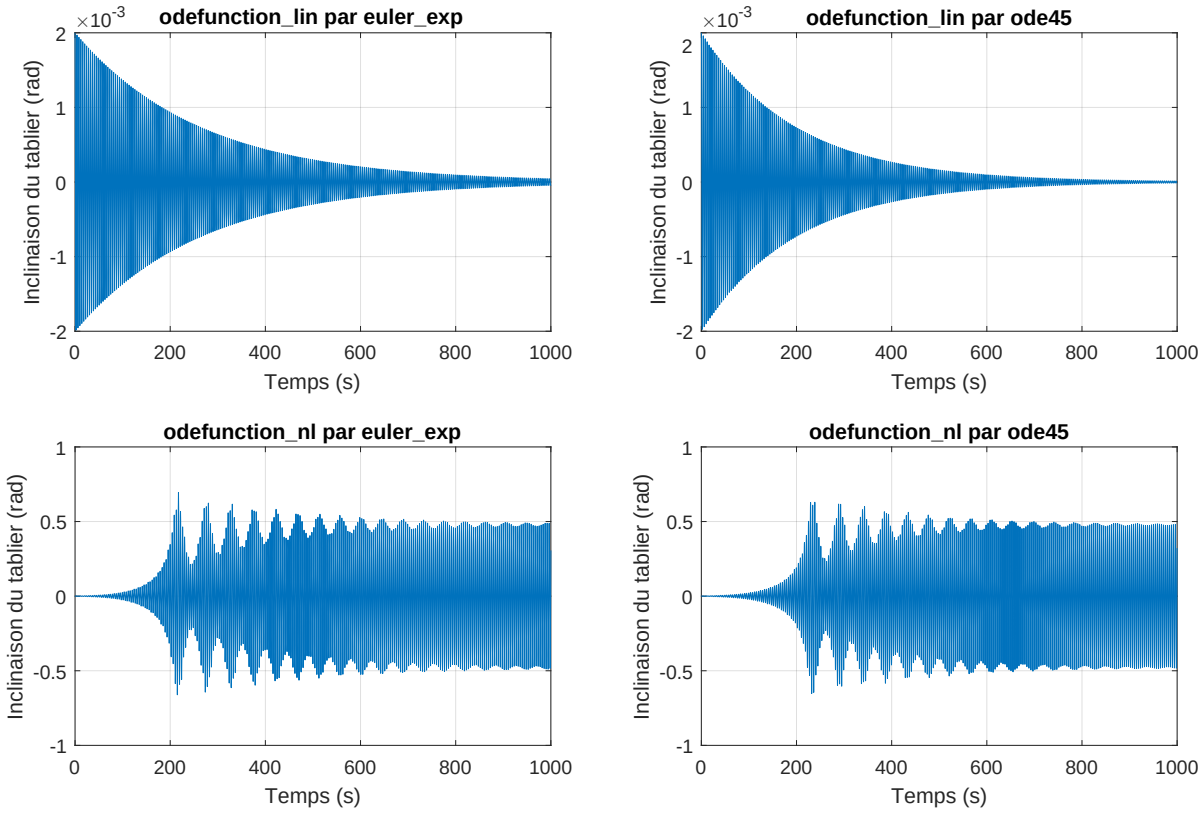

Figure 3 – Variation de l'angle de torsion pour les modèles linéaire et non-linéaire, résolus avec les méthodes Euler explicite et `ode45`

Au vu des résultats obtenus dans cet exemple-ci, nous considérerons dans le reste du problème la méthode de *Runge-Kunta* avec l'un de ses solveurs pour son efficacité et sa robustesse. En ce qui concerne le modèle, nous représenterons le comportement du pont à l'aide du non-linéaire, les simplifications du modèle linéaire limitant la fiabilité des résultats.

3 Étude de la stabilité du tablier de pont

Dans cette dernière partie du problème, il nous est demandé de déterminer et d'étudier, sous différents paramètres, l'amplitude de la sollicitation cyclique verticale due au vent qui mènerait à la rupture du pont. Dans un soucis de réalisme, seul le modèle non-linéaire sera sujet à cette étude. Cependant, lors de nos calculs, nous avons pu constater que le système d'équations différentielles fourni était raide, ce qui rend très difficile sa résolution par des méthodes numériques. Sous conseil des assistants, nous avons donc préféré à la fonction `ode45`, la fonction `ode15s` plus à même de gérer un système raide.

3.1 Tension maximale exercée dans les câbles

Sous l'action du vent, des tensions s'exercent dans les câbles soutenant le pont. Il nous est demandé de déterminer la tension maximale exercée dans ceux-ci pour une amplitude de 27 500 N et un paramètre d'amortissement des câbles de 0,1 m^{-1}.

Pour obtenir le maximum demandé, nous avons calculé les tensions s'exerçant dans les deux câbles pour chaque instant déterminé par `ode15s`. Nous avons ensuite isolé la tension maximale de chacun des deux câbles pour ne reprendre finalement que la plus grande des deux. Mais, notre opération s'est effectuée sur un intervalle discontinu. Pour nous approcher de la réalité, nous avons repris un certain nombre de valeurs autour de celle trouvée et nous avons effectué une interpolation sur celles-ci. Cela nous a permis de trouver un second maximum plus réaliste. Après implémentation, nous trouvons une tension maximale de 7533,384 N.

3.2 Étude de l'amplitude de la sollicitation cyclique verticale due au vent

Nous devons étudier l'amplitude de la sollicitation cyclique verticale due au vent qui mènerait à la rupture du pont. Par soucis de simplicité, nous appellerons cette amplitude « l'amplitude de rupture ».

3.2.1 Détermination de l'amplitude de rupture

Nous savons que si une tension de 7900 N s'exerce dans un des deux câbles du pont, cela conduirait à sa rupture. Il nous est maintenant demandé de déterminer à quelle amplitude correspond une tension maximale de 7900 N dans un des câbles.

L'amplitude était, à la base, un paramètre fixe du système d'équations différentielles. Il nous faut désormais le considérer comme un paramètre variable.

À chaque itération, la position du tablier et la tension maximale dans les câbles sont recalculées. Nous obtenons donc différentes valeurs de tensions maximales associées à des valeurs de l'amplitude. Il nous reste à utiliser la méthode de la bissection pour trouver à quelle amplitude correspond une tension maximale de 7900 N.

Après implémentation et avec une tolérance de une unité pour les ordonnées et les abscisses, nous obtenons une amplitude de rupture de 28 810 N. Pour des chiffres si élevés, il n'est pas nécessaire d'augmenter plus notre précision.

3.2.2 Évolution de l'amplitude du vent

L'amplitude de rupture étant déterminée pour les conditions données, il nous est demandé d'étudier son évolution lorsque le paramètre α varie. Pour cela, nous reprenons l'algorithme implémenté précédemment, et nous la re-calculons pour des valeurs de α évoluant de 0 à 0,3 rad, par pas de 0,000 75 rad.

Nous avons obtenu le résultat suivant : plus le paramètre α augmente, plus l'amplitude de rupture diminue. En observant le graphique, on remarque que la relation entre l'amplitude et α est presque anti-proportionnelle. Cependant, la courbe présente de légères oscillations. Une explication plausible est qu'il y aurait plusieurs amplitudes répondant aux critères. Ainsi, notre bissection ne trouverait pas toujours la plus faible de ces valeurs ce qui induirait les cassures.

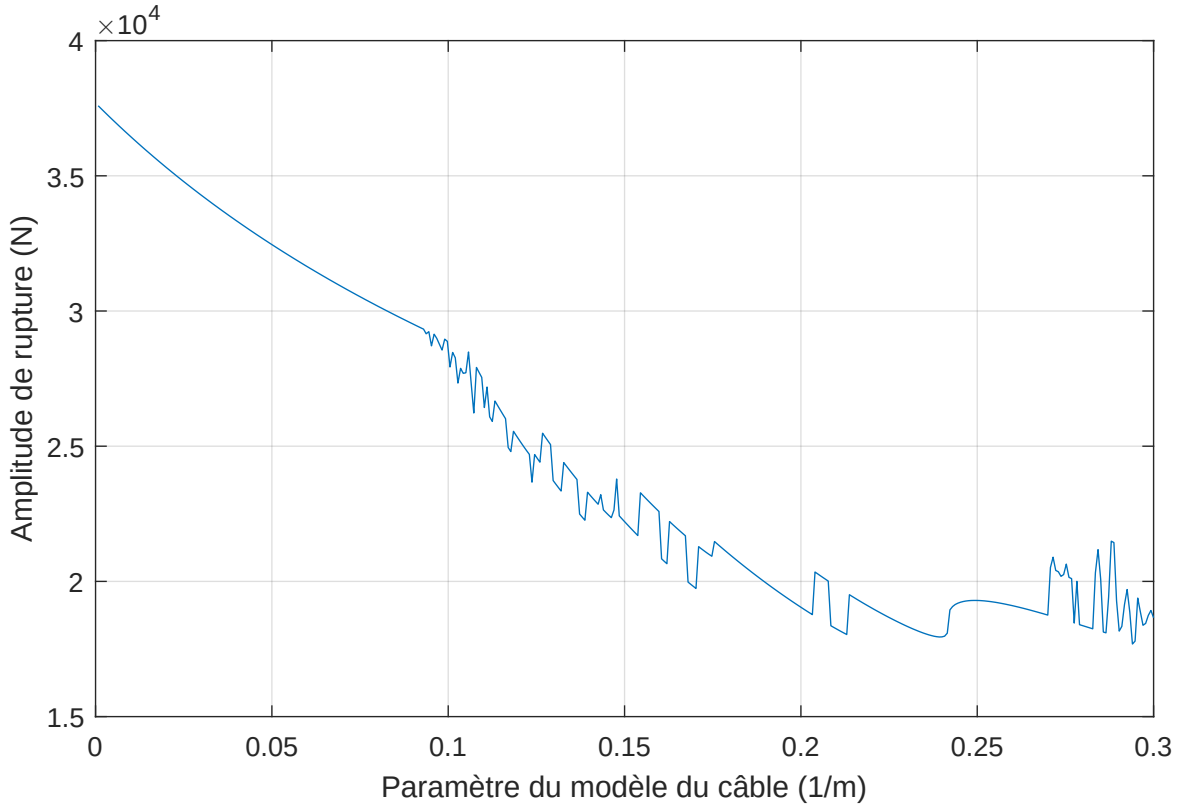

Figure 4 – Évolution de l'amplitude de rupture par rapport au paramètre α, variant de 0 à 0,3 rad par pas de 0,000 75 rad

3.2.3 Choix des intervalles, tolérances et précisions

Dans ce dernier problème, nous utilisons de nouveau la méthode de la bissection pour sa robustesse. Nous devons alors re-définir nos points de départ x_0 et x_1, et les tolérances sur les abscisses et ordonnées.

En ce qui concerne les points de départ, nous avons choisi 0 et 40 000 N. Le point 0 est l'amplitude minimum que nous pourrions avoir, et le point 40 000 majore l'amplitude maximum qui pourrait être atteinte (déterminée empiriquement). Toutes les amplitudes calculées se trouveront donc forcément dans cet intervalle, et nos points seront donc bien de part et d'autre de l'abscisse recherchée.

En ce qui concerne les tolérances, nous avons pris 1 pour les abscisses et les ordonnées. Les valeurs de base étant données avec une précision à l'unité, il nous a semblé opportun

de conserver celle-ci pour fournir notre réponse.

Nous utilisons également la fonction `ode15s`. Tout comme `ode45`, nous devons définir `AbsTol` et `RelTol`. Pour ce problème-ci, nous avons choisi un `AbsTol` de 10^{-6} et un `RelTol` de 10^{-7}. La différence des valeurs calculées avec des `RelTol` inférieurs est nulle, ce qui nous pousse à ne pas descendre plus bas que 10^{-7}.

4 Sources consultées

4.1 Bibliographie

1. Professeur Louveaux Q., *Introduction aux méthodes numériques*, 2016-2017
2. Professeur Denoël V., *Stabilité d'un pont suspendu soumis à des sollicitations cycliques - Mise en contexte*, 2017
3. Moore H., *MATLAB for Engineers - Third Edition*, 2012
4. The MathWorks, Inc., *MATLAB - Object-Oriented Programming*, 2015

4.2 Webographie

1. Documentation MATLAB, `https://nl.mathworks.com/help/matlab`, consultée en février, mars et avril 2017
2. Documentation `ode`, `https://nl.mathworks.com/help/matlab/math/choose-an-ode-solver.html`, consultée en février, mars et avril 2017
3. Documentation LaTeX, `https://fr.wikibooks.org/wiki/LaTeX`, consultée en février, mars et avril 2017
4. Documentation `parfor`, `https://nl.mathworks.com/help/distcomp/parfor.html`, consultée en avril 2017
5. Documentation `try...catch`, `https://nl.mathworks.com/help/matlab/ref/try.html`, consultée en mars et avril 2017