Compte rendu : TP optimisation
Equilibrage d'une chaîne

Réalisé par : Mohamed ISSA

Classe : $2^{ère}$ Année Techniques Avancées

Année universitaire : 2020/2021

Table des matières

Table des figures

1 Position du problème

Dans ce TP on se propose de déterminer la position d'équilibre statique d'une chaîne constituée de n_b barres rigides contenue dans un plan vertical (x, y). Les variables du problème sont les coordonnées des noeuds n_i d'articulation de la chaîne sans prendre en compte les extrémités de la chaîne. La figure 1 montre une position quelconque de la chaîne d'extrémités $(0, 0)$ et (a, b).

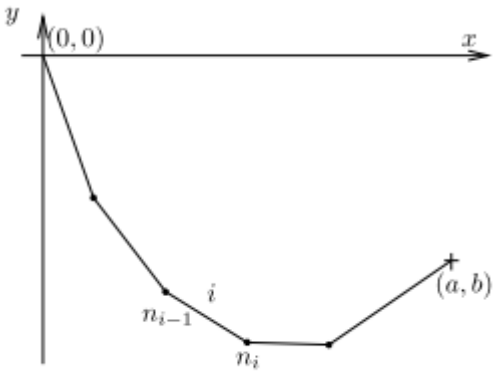

FIGURE 1 – chaîne formée de barres.

Pour ce faire, on minimise l'énergie potentielle de la barre sous les contraintes d'égalités qui font correspondre les longueurs l_i entre les noeuds aux données du problèmes L_i c-à-d $l_i = L_i$ pour cela la contrainte d'égalité c_i s'écrit $c_i = l_i^2 - L_i^2 = 0$.

En résumé le problème à résoudre est le problème de minimisation ci-dessous :

$$(P_1) \begin{cases} min\, e(x, y) \\ c_i(x, y) = 0 \quad i = 1...n_b \end{cases}$$

où

$$e(x, y) = \sum_{i=1}^{n_b} L_i \frac{y_i + y_{i+1}}{2}$$

2 Méthode de résolution

La méthode de résolution de ce problème consiste à chercher les points stationnaires (x_*, λ_*) du système d'optimalité ci-dessous :

$$(S) : \begin{cases} \nabla e(x_*) + c'(x_*)^T \lambda_* = 0 \\ c(x_*) = 0 \end{cases}$$

Pour résoudre ce système on utilise la méthode de **l'algorithme de Newton** pour les systèmes non linéaires pour la fonction F définie par les deux gradients du Lagrangien pour les variables x et λ.

On cherche $z_* = (x_*, \lambda_*)$ tq $F(z_*) = 0$ avec :

$$F(z) = \begin{pmatrix} \nabla e(x) + c'(x)^T \lambda \\ c(x) \end{pmatrix}, \; z = (x, \lambda) \in \mathbf{R}^N$$

Le système obtenu en appliquant l'algorithme de Newton revient à :

$$\begin{pmatrix} L_k & A_k^T \\ A_k & 0 \end{pmatrix} \begin{pmatrix} d_k \\ \lambda_k^{PQ} \end{pmatrix} = - \begin{pmatrix} \nabla e(x_k) \\ c(x_k)) \end{pmatrix}$$

avec :

$$L_k = \nabla_{xx} l(x_k, \lambda_k) \text{ et } A_k = c'(x_k)$$

Les noveaux itérés sont donnés par :

$$x_{k+1} = x_k + d_k \text{ et } \lambda_{k+1} = \lambda_k^{PQ}$$

3 Implémentation

L'implémentation du code respecte l'architecture de la communication directe d'un problème d'optimisation. La figure 2 montre les différentes routines du projet organisées selon cette structure.

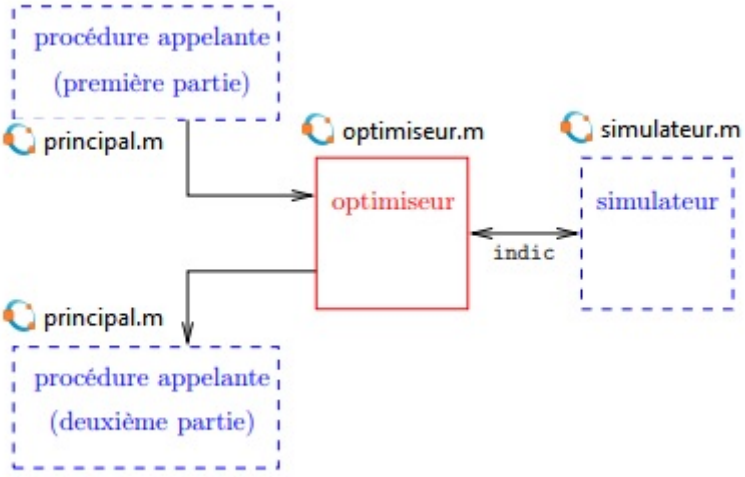

FIGURE 2 – Schéma du code d'optimisation en communication directe.

3.1 Procédure appelante

C'est le programme principal où sont citées les différentes données du problème. Celui-ci fait appel à l'optimiseur pour calculer une éventuelle solution du problème. L'appel se fait par l'instruction suivante :

```
[xy,lm,info] = optimiseur('simulateur',xy,lm,options);
```

FIGURE 3 – Appel de l'optimiseur.

3.2 Optimiseur

Le rôle de l'optimiseur est de chercher une solution du problème en appliquant l'algorithme de Newton. Il communique avec le simulateur. L'entête de l'optimiseur est donnée dans la figure 4.

Les arguments de la fonction *optimiseur.h* sont :
— simul :'simulateur' qui est le nom du simulateur ;
— xy : c'est un point de départ de l'algorithme ;
— lm : multiplicateur de départ de l'algorithme ;
— options : c'est une structure dont les attributs *options.tol* est un vecteur donnant les seuils de tolérance *tol(1)* et *tol(2)* et *options.maxit* qui est le nombre maximal d'itérations.

```
function [xy,lm,info] = optimiseur (simul,xy,lm,options)
```

FIGURE 4 – entête de la fonction optimiseur.h

Cette fonction retourne une solution (xy, lm) et $info$ qui vaut 0 si le seuil d'optimalité est atteint et 1 si le nombre maximal d'itérations est atteint.

3.3 Simulateur

Le simulateur *simulateur.h* est appelé par l'optimiseur à chaque fois il a besoin d'un calcul de dérivé ou d'une donnée quelconque du problème.L'entête de la foncton *simulateur.h* est donnée dans la figure 5. Les données de la fonction *simulateur.h* sont :
— xy : les coordonnées des noeuds ;
— lm : un multiplicateur donné ;
— indic : s'il vaut 1, la fonction trace la chaîne. S'il vaut 2, elle retourne e et c. S'il vaut 4, elle retourn e, c, g et a. Et s'il vaut 5 elle calcule hl.

```
function [e,c,g,a,hl,indic] = simulateur (indic,xy,lm)
```

FIGURE 5 – entête de la fonction simulateur.h

4 Tests et interprétations

4.1 Cas-test 1

On saisit dans le programme principal les données du problème comme le montre la figure 6 et on fait à chaque fois varier la valeur de l'indice *indic* pour déterminer les valeurs des différentes sorties. La figure 7 montre la figure obtenue avec $indic = 1$. Les valeurs de e, c, g et a sont données dans la figure 9 et la valeur de hl pour le multiplicateur lm (voir 6) est donnée dans la figure 8.

```
global A B L
A=  1;
B= -1;
L=[0.7 0.5 0.3 0.2 0.5]';
xy = [0.2 0.4 0.6 0.8 ...
      -1.0 -1.5 -1.5 -1.3 ]';

lm= [0.5077 0.4223 ...
0.5190 0.6156 0.8774]';

[e,c,g,a,hl,indic]=simulateur(1,xy,lm);
```

FIGURE 6 – Données du premier test.

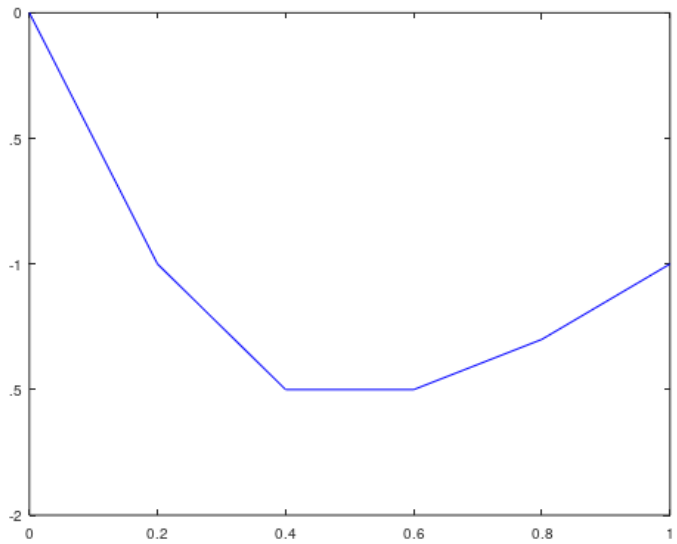

FIGURE 7 – figure de la chaîne pour le test 1.

On fixe les tolérances pour une valeur de 0.01 et le nombre maximal d'itérations à 14 dans le programme principal et on exécute le programme. L'algorithme converge en 5 itérations et la figure 10 montre les différentes points au cours des itérations.

Au cours des itérations, j'ai constaté une augmentation de la valeur de e (voir figure 11).

```
>> hl
hl =

   1.86000  -0.84460   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  -0.84460   1.88260  -1.03800   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -1.03800   2.26920  -1.23120   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -1.23120   2.98600   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.86000  -0.84460   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -0.84460   1.88260  -1.03800   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -1.03800   2.26920  -1.23120
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -1.23120   2.98600
```

FIGURE 8 – hl pour le test 1.

Pour le cas du test 1 et afin de tracer la courbe de :

$$r(k) = \frac{||\nabla_x l(x_k, \lambda_k)||_\infty + ||c(x_k)||_\infty}{||\nabla_x l(x_1, \lambda_1)||_\infty + ||c(x_1)||_\infty}$$

j'ai ajouté aux arguments de sorties de l'optimiseur deux tableaux *gl* et *cc*. *gl* contient les valeurs de $||\nabla_x l(x_k, \lambda_k)||_\infty$ et *cc* contient les valeurs de $||c(x_k)||_\infty$.

Les valeurs trouvées sont données dans la figure 12.

La courbe à l'échelle logarithmique $log(r)$ en fontion de $log(k)$ est donnée par la figure 13.

On remarque que cette courbe est décroissante ce qui montre la convergence de l'algorithme de Newton pour le test 1. On peut déduire même que la convergence est quadratique pour l'algorithme de Newton.

```
>> principal

e = -2.2800
>> c
c =

   0.550000
   0.040000
  -0.050000
   0.040000
  -0.120000

>> g
g =

   0.00000
   0.00000
   0.00000
   0.00000
   0.60000
   0.40000
   0.25000
   0.35000

>> a
a =

   0.40000   0.00000   0.00000   0.00000  -2.00000   0.00000   0.00000   0.00000
  -0.40000   0.40000   0.00000   0.00000   1.00000  -1.00000   0.00000   0.00000
   0.00000  -0.40000   0.40000   0.00000   0.00000  -0.00000   0.00000   0.00000
   0.00000   0.00000  -0.40000   0.40000   0.00000   0.00000  -0.40000   0.40000
   0.00000   0.00000   0.00000  -0.40000   0.00000   0.00000   0.00000  -0.60000
```

FIGURE 9 – Les valeurs de e,c,g et a pour le test 1.

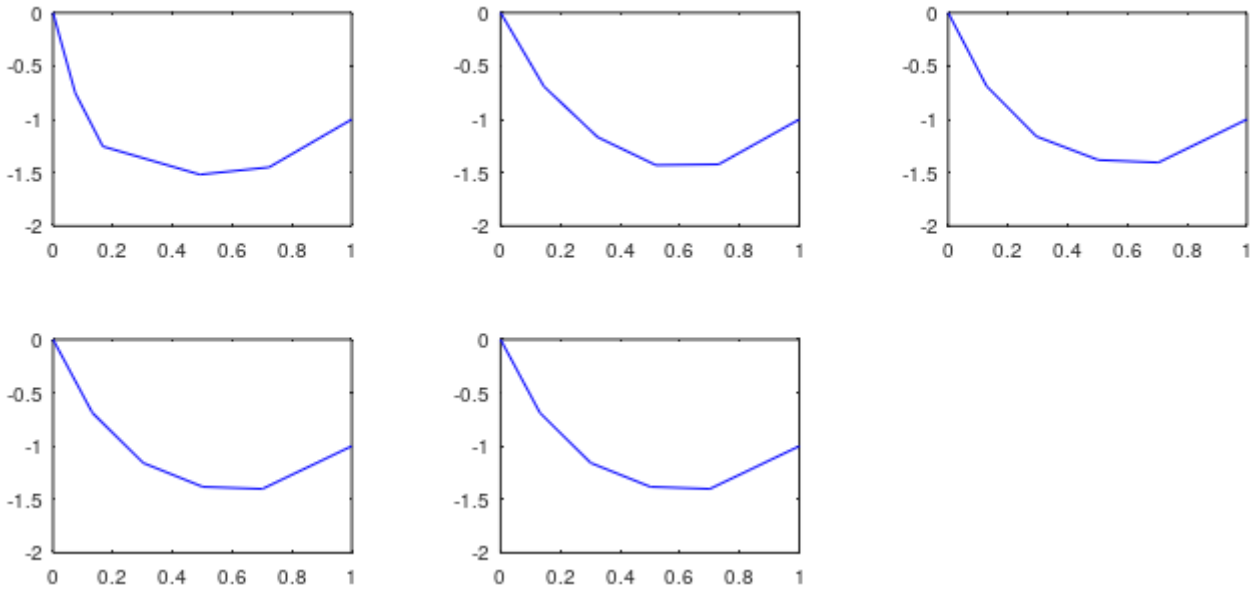

FIGURE 10 – Positions des noeuds au cours des itérations (test 1).

```
e = -2.2800
e = -2.0878
e = -1.9860
e = -1.9636
e = -1.9612
```

FIGURE 11 – Valeurs de e au cours des itérations (test 1).

```
>> gl
gl =

   0.10472000   0.30020609   0.07283047   0.01763376   0.00033918

>> cc
cc =

   0.550000000   0.085243641   0.016907419   0.001883011   0.000072580
```

FIGURE 12 – Les valeurs de gl et cc.

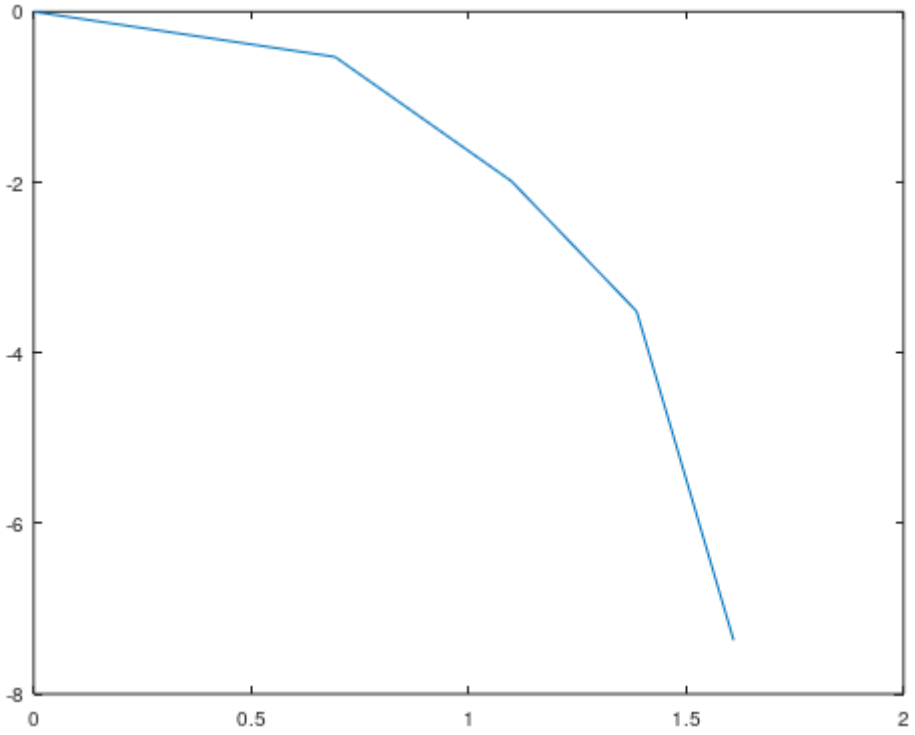

FIGURE 13 – courbe logarithmique de r en fonction de k.

4.2 Cas-test 1b

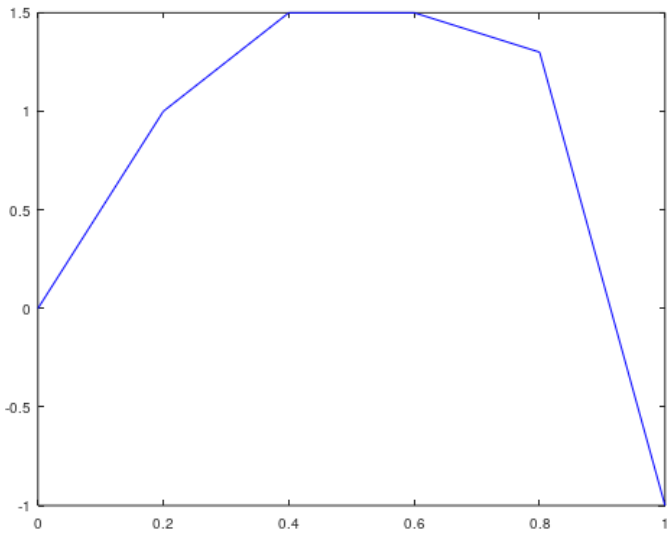

FIGURE 14 – figure de la chaîne pour le test 1b.

```
>> hl
hl =

   1.86000  -0.84460   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  -0.84460   1.88260  -1.03800   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -1.03800   2.26920  -1.23120   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -1.23120   2.98600   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.86000  -0.84460   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -0.84460   1.88260  -1.03800   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -1.03800   2.26920  -1.23120
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -1.23120   2.98600
```

FIGURE 15 – hl pour le test 1b.

L'algorithme converge au bout de 11 itérations. La figure 17 montre les positions des points au cours des itérations.

On remarque une diminution de la valeur de e aux dernières itérations (voir 18).

```
>> e
e =  1.7800
>> c
c =

   0.550000
   0.040000
  -0.050000
   0.040000
   5.080000

>> g
g =

   0.00000
   0.00000
   0.00000
   0.00000
   0.60000
   0.40000
   0.25000
   0.35000

>> a
a =

   0.40000   0.00000   0.00000   0.00000   2.00000   0.00000   0.00000   0.00000
  -0.40000   0.40000   0.00000   0.00000  -1.00000   1.00000   0.00000   0.00000
   0.00000  -0.40000   0.40000   0.00000   0.00000  -0.00000   0.00000   0.00000
   0.00000   0.00000  -0.40000   0.40000   0.00000   0.00000   0.40000  -0.40000
   0.00000   0.00000   0.00000  -0.40000   0.00000   0.00000   0.00000   4.60000
```

FIGURE 16 – Les valeurs de e,c,g et a pour le test 1b.

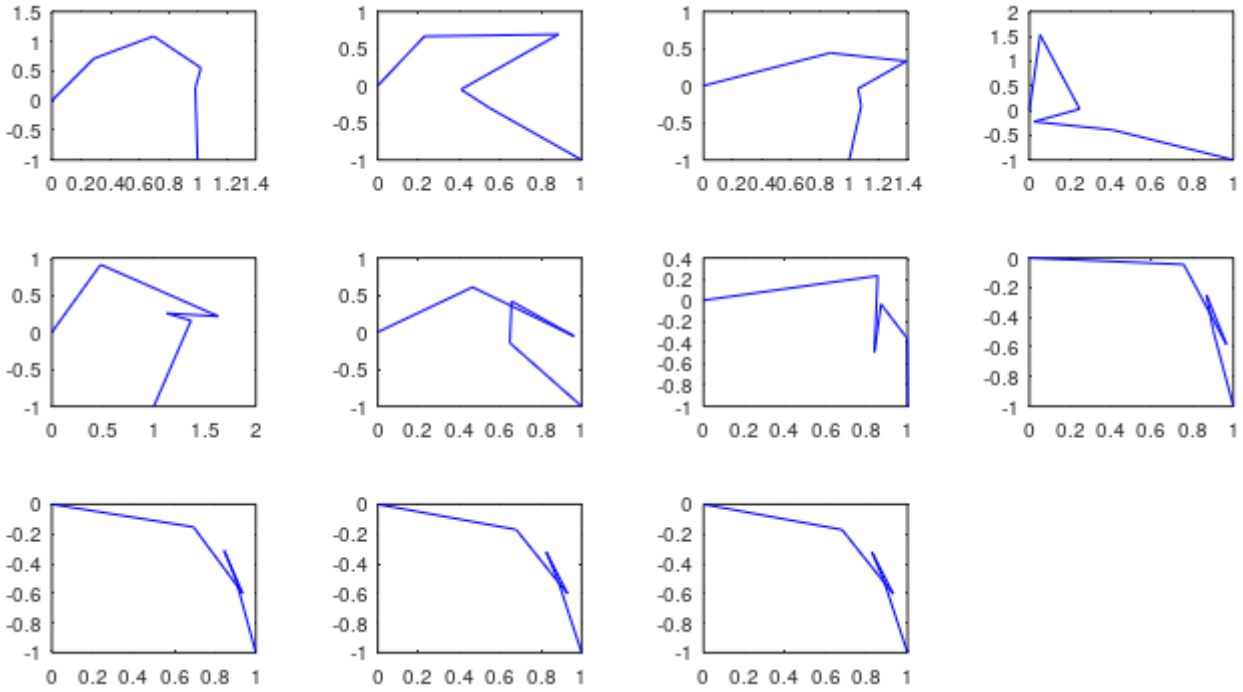

FIGURE 17 – Positions des noeuds au cours des itérations (test 1b).

```
e =  1.7800
e =  0.81947
e =  0.30958
e =  0.041747
e =  0.48425
e =  0.49787
e =  0.14406
e = -0.44056
e = -0.74176
e = -0.83853
e = -0.85388
```

FIGURE 18 – valeurs de e au cours des itérations (test 1b).

4.3 Cas-test 1c

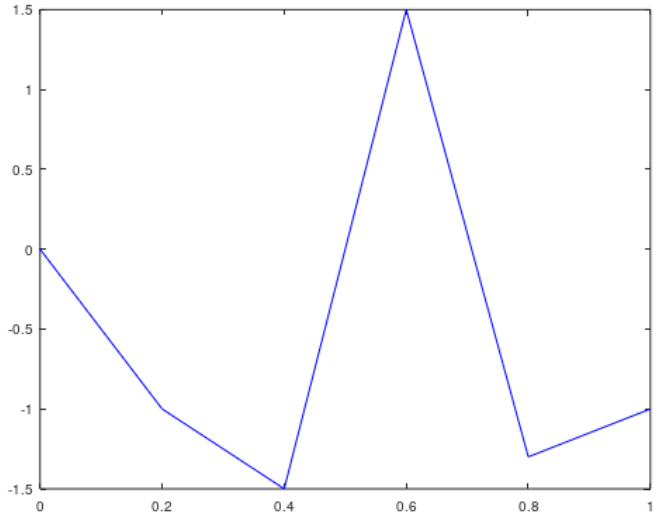

FIGURE 19 – figure de la chaîne pour le test 1c.

```
>> hl
hl =

   1.86000  -0.84460   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  -0.84460   1.88260  -1.03800   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -1.03800   2.26920  -1.23120   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -1.23120   2.98600   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.86000  -0.84460   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -0.84460   1.88260  -1.03800   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -1.03800   2.26920  -1.23120
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -1.23120   2.98600
```

FIGURE 20 – hl pour le test 1c.

L'algorithme converge au bout de 11 itérations. La figure 22 montre les positions au cours des itérations.

On remarque que les valeurs prises par e (voir fig 23) ne diminuent pas strictement et n'augmentent pas strictement au voisinage du point stationnaire.

```
>> e
e = -1.5300
>> c
c =

   0.550000
   0.040000
   8.950000
   7.840000
  -0.120000

>> g
g =

   0.00000
   0.00000
   0.00000
   0.00000
   0.60000
   0.40000
   0.25000
   0.35000

>> a
a =

   0.40000   0.00000   0.00000   0.00000  -2.00000   0.00000   0.00000   0.00000
  -0.40000   0.40000   0.00000   0.00000   1.00000  -1.00000   0.00000   0.00000
   0.00000  -0.40000   0.40000   0.00000   0.00000  -6.00000   6.00000   0.00000
   0.00000   0.00000  -0.40000   0.40000   0.00000   0.00000   5.60000  -5.60000
   0.00000   0.00000   0.00000  -0.40000   0.00000   0.00000   0.00000  -0.60000
```

FIGURE 21 – Les valeurs de e,c,g et a pour le test 1c.

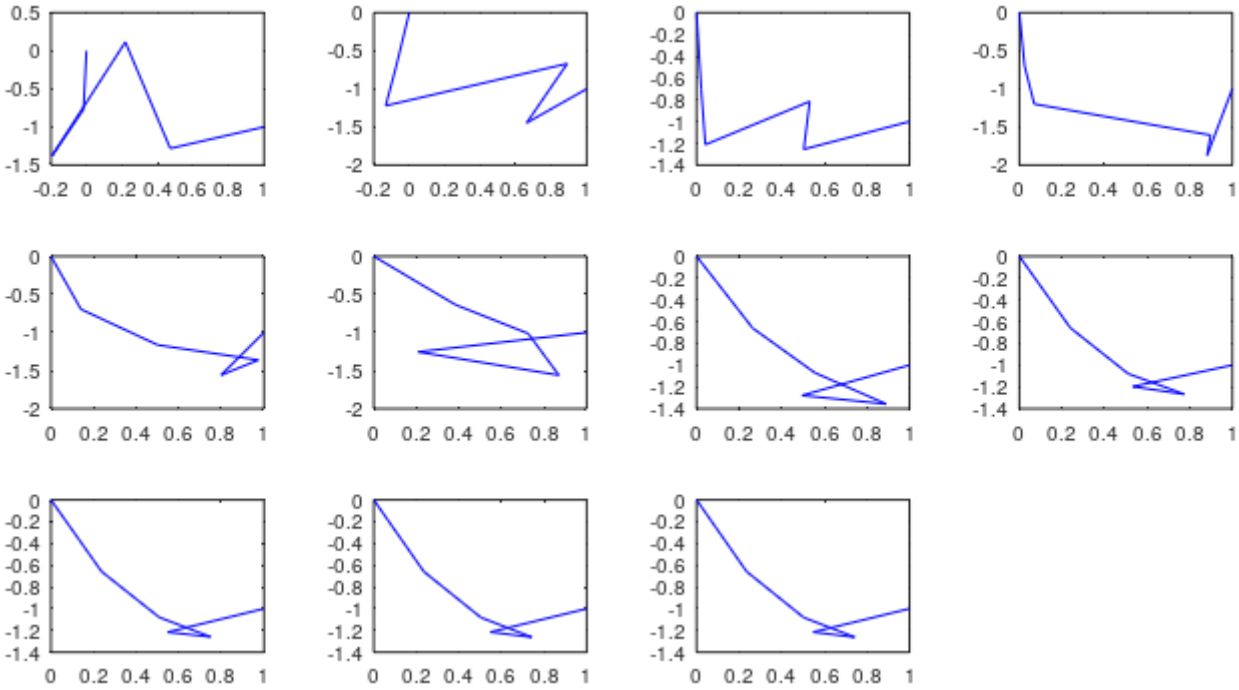

FIGURE 22 – Positions des noeuds au cours des itérations (test 1c)

```
e = -1.5300
e = -1.6826
e = -1.8323
e = -1.8016
e = -2.2057
e = -2.0196
e = -1.8669
e = -1.8609
e = -1.8123
e = -1.8168
e = -1.8187
```

FIGURE 23 – valeurs de e au cours des itérations (test 1c).

4.4 Cas-test 1d

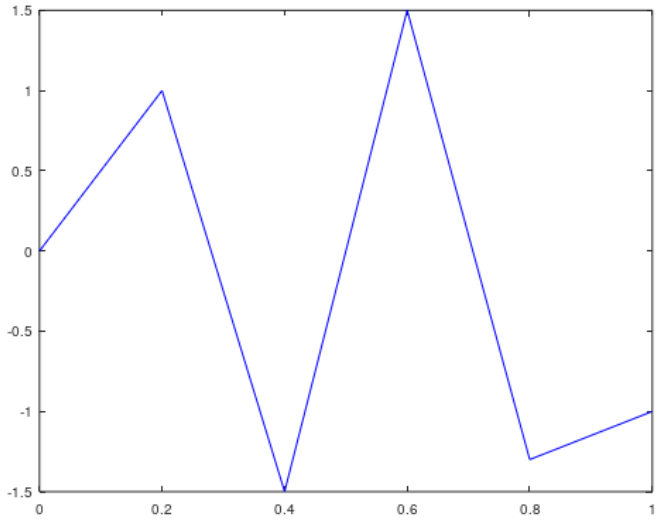

FIGURE 24 – figure de la chaîne pour le test 1d.

```
>> hl
hl =

   1.86000  -0.84460   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000
  -0.84460   1.88260  -1.03800   0.00000   0.00000   0.00000   0.00000   0.00000
   0.00000  -1.03800   2.26920  -1.23120   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000  -1.23120   2.98600   0.00000   0.00000   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000   1.86000  -0.84460   0.00000   0.00000
   0.00000   0.00000   0.00000   0.00000  -0.84460   1.88260  -1.03800   0.00000
   0.00000   0.00000   0.00000   0.00000   0.00000  -1.03800   2.26920  -1.23120
   0.00000   0.00000   0.00000   0.00000   0.00000   0.00000  -1.23120   2.98600
```

FIGURE 25 – hl pour le test 1d.

L'algorithme ne converge pas. On peut voir que $info.status$ a pris la valeur 2 (voir fig28) ce qui signifie que l'algorithme a atteint la valeur maximale des itérations. La figure 27 montre les positions au cours des itérations.

Les valeurs de e (fig 28) varient fortement ce qui est justifié par la non-convergence de l'algorithme.

```
>> e
e = -0.33000
>> c
c =

   0.55000
   6.04000
   8.95000
   7.84000
  -0.12000

>> g
g =

   0.00000
   0.00000
   0.00000
   0.00000
   0.60000
   0.40000
   0.25000
   0.35000

>> a
a =

   0.40000   0.00000   0.00000   0.00000   2.00000   0.00000   0.00000   0.00000
  -0.40000   0.40000   0.00000   0.00000   5.00000  -5.00000   0.00000   0.00000
   0.00000  -0.40000   0.40000   0.00000   0.00000  -6.00000   6.00000   0.00000
   0.00000   0.00000  -0.40000   0.40000   0.00000   0.00000   5.60000  -5.60000
   0.00000   0.00000   0.00000  -0.40000   0.00000   0.00000   0.00000  -0.60000
```

FIGURE 26 – Les valeurs de e,c,g et a pour le test 1d.

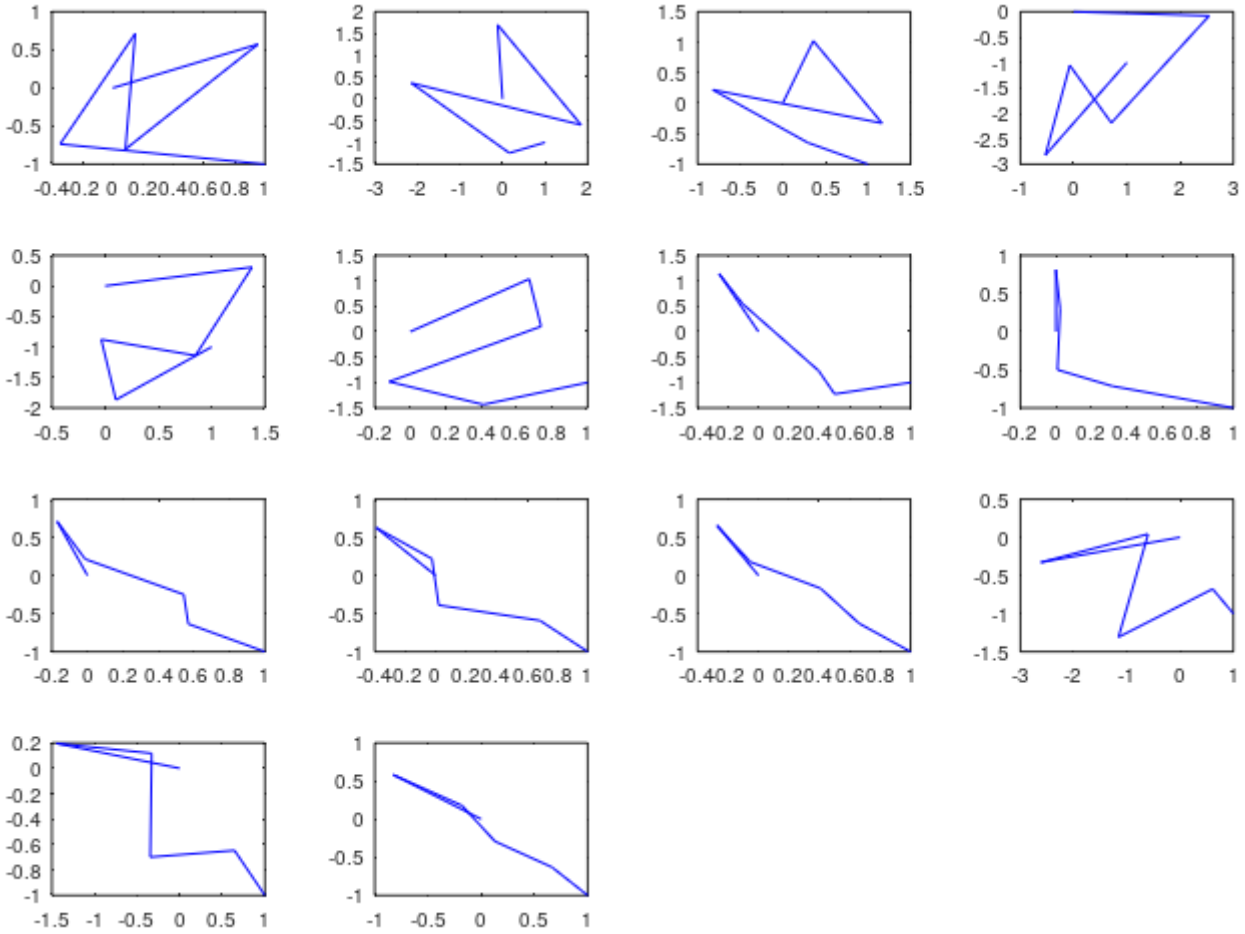

FIGURE 27 – Positions des noeuds au cours des itérations (test 1d)

```
e = -0.33000
e = -0.30628
e =  0.18645
e =  0.060820
e = -2.4247
e = -1.3945
e = -0.33468
e =  0.030050
e = -0.019546
e = -0.024524
e = -0.091226
e = -0.047616
e = -0.99449
e = -0.48581
>> info.status
ans =  2
```

FIGURE 28 – valeurs de e au cours des itérations et info.status (test 1c).