HAL open science

Résolution de petits systèmes algébriques par la MAN sous Matlab

Bruno Cochelin, Franck Pérignon

▶ To cite this version:

Bruno Cochelin, Franck Pérignon. Résolution de petits systèmes algébriques par la MAN sous Matlab. Revue Européenne des Éléments Finis, 2004, 13, pp.79-96. 10.3166/reef.13.79-96 . hal-00089039

HAL Id: hal-00089039

https://hal.science/hal-00089039v1

Submitted on 9 Aug 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Résolution de petits systèmes algébriques par la MAN sous Matlab

Bruno Cochelin — Franck Pérignon

Laboratoire de Mécanique et d'Acoustique CNRS UPR 7051
Ecole d'Ingénieurs Généralistes de Marseille
Technopôle de Château Gombert
F-13383 Marseille Cedex 20

{cochelin,perignon}@egim-mrs.fr

RÉSUMÉ. *Dans cet article, nous présentons un logiciel de continuation destiné à résoudre des systèmes algébriques non linéaires dépendant de paramètres. Nous rappelons le principe de continuation d'une branche par une méthode asymptotique numérique (MAN) en nous appuyant sur un exemple simple, et en détaillant progressivement les étapes du programme. Nous traitons ensuite du calcul d'un premier point solution et de la façon d'effectuer des sauts d'une branche à l'autre. Quelques exemples complémentaires sont ensuite proposés pour illustrer une partie essentielle des MAN : l'écriture des équations sous une forme quadratique.*

ABSTRACT. *This paper presents a Matlab software for the continuation of solutions of algebraic systems with parameters. We recall the principle of continuation with the MAN with an example, and we progressively introduce all the routines of the program. Next, we discuss the problem of finding a first solution point on the branch, and the problem of jumping onto another branch. A few additional examples are given to insist on a crucial point : writing the equation in a quadratic framework.*

MOTS-CLÉS : *Continuation, Méthode asymptotique numérique, Matlab.*

KEYWORDS: *Continuation, Asymptotic numerical method, Matlab.*

1. Introduction

La MAN (méthode asymptotique numérique) est une technique de résolution non linéaire qui a été développée au début des années 90 pour effectuer la continuation des branches de solutions pour les problèmes de flambement de structures [DAM 90, AZR 93, COC 94]. De nombreux aménagements et améliorations ont été proposés depuis pour étendre le domaine d'application à d'autres problèmes de mécanique et de physique, comme le montre la diversité des thèmes abordés dans ce numéro spécial de la Revue Européenne des Eléments Finis. La MAN est aujourd'hui un *solveur* non linéaire assez général permettant de résoudre les systèmes algébriques issus d'une équation aux dérivées partielles et d'une méthode de discrétisation (éléments finis, différences finies, volumes finis...).

Par rapport aux autres techniques de continuation, et notamment les méthodes incrémentales-itératives [KEL 87, ALL 90, CRI 97], les avantages et les inconvénients de la MAN sont maintenant bien cernés. Il est établi que la très grande robustesse de continuation est due aux représentations analytiques des branches de solutions sous la forme de séries entières tronquées à des ordres élevés. Ces séries sont en effet très riches en informations locales sur le tronçon de branche que l'on cherche à déterminer, et elles permettent de gérer efficacement le pilotage de la continuation, la longueur des pas, et la détection des bifurcations par exemple. C'est pour ces raisons que, même pour les petits systèmes algébriques, où les temps de calcul ne sont pas un obstacle *a priori*, la MAN peut constituer une alternative intéressante aux méthodes de continuation standards.

Dans cet article, nous présentons un logiciel de continuation destiné à résoudre les petits[1] systèmes algébriques dépendant de paramètres. Nous n'introduirons pas de concepts nouveaux sur la MAN, mais nous nous attacherons à détailler les algorithmes et leur mise en œuvre dans le logiciel Matlab. Nous montrerons à cette occasion que la programmation d'une MAN est assez élémentaire : le cœur du programme (calcul des séries) tient sur une quinzaine de lignes, et l'ensemble du logiciel de continuation en une petite dizaine de fonctions et moins de 100 lignes. Nous avons choisi de nous appuyer sur un exemple simple de continuation, en donnant les fragements de théorie et les programmes au fur et à mesure. A la fin du papier, nous reviendrons sur une partie essentielle qui est la transformation du système de départ en un système d'équations quadratiques. Il en découle une question d'importance sur les limites de la MAN : « peut on appliquer la MAN à n'importe quel système d'équations algébriques ? ».

2. Continuation d'une branche de solution

On considère un système non linéaire algébrique de N_{eq} équations à N_{eq} inconnues et un paramètre λ.

$$r(u,\lambda) = 0 \quad u \in R^{N_{eq}}, r \in R^{N_{eq}}, \lambda \in R \qquad [1]$$

1. On pense ici à quelques dizaines ou quelques centaines d'équations.

Soit u_0,λ_0 un point solution de [1] et u_1,λ_1 le vecteur tangent en ce point. Pour effectuer la continuation de la branche de solutions qui part de u_0,λ_0, selon la direction donnée par u_1,λ_1, on en détermine une représentation locale sous la forme de séries entières tronquées [2] à un ordre élevé ($N = 20$ ou 30 en pratique) :

$$\begin{aligned} u(a) &= u_0 + au_1 + a^2u_2 + a^3u_3 + \ldots + a^N u_N \\ \lambda(a) &= \lambda_0 + a\lambda_1 + a^2\lambda_2 + a^3\lambda_3 + \ldots + a^N \lambda_N \end{aligned} \qquad [2]$$

où a est un paramètre de chemin. Les séries [2] ont en général un rayon de convergence fini et elles ne décrivent de façon précise qu'un certain tronçon de la branche au voisinage de u_0,λ_0. Pour progresser sur la branche, il faut définir un nouveau point de départ sur le tronçon déjà calculé, et évaluer à nouveau les séries pour obtenir un nouveau tronçon de branche, comme cela est montré sur le figure 1. En procédant ainsi, on obtient une représentation complète de la branche sous la forme d'une succession de séries entières. Nous allons maintenant rentrer dans les détails en nous appuyant sur un exemple.

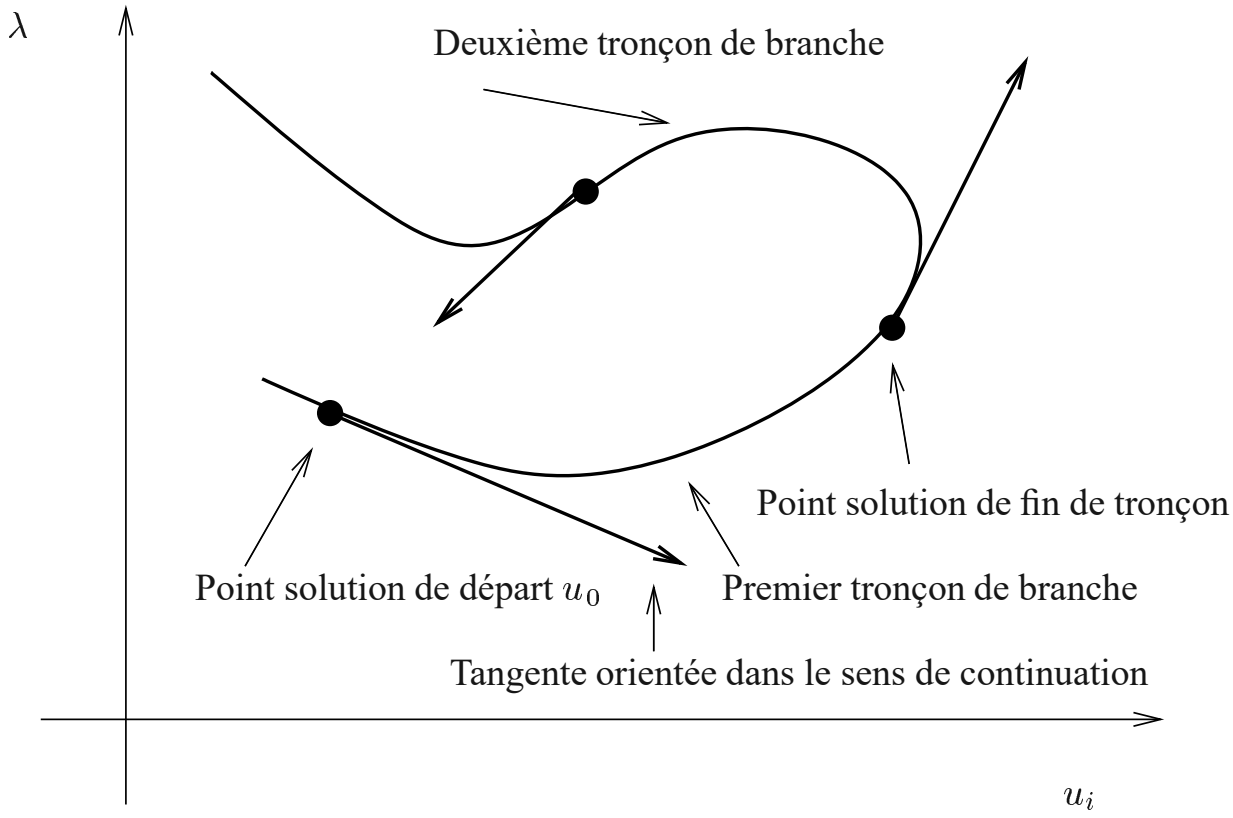

Figure 1. *Continuation d'une branche par MAN : la branche est déterminée comme une succession de tronçons. Chaque tronçon correpond à un développement en séries de type [2]*

2. Nous verrons plus loin comment calculer ces séries.

2.1. *L'exemple retenu*

Nous reprenons l'exemple de réaction bio-chimique utilisé dans [DOE 91] pour illustrer les méthodes de continuation. Les inconnues sont les concentrations u_1 et u_2 d'un produit réactif dans des compartiments qui autorisent des échanges entre eux et avec l'extérieur. Le paramètre λ est la concentration du milieu extérieur et μ est un paramètre perturbateur. Les équations qui gouvernent les solutions stationnaires d'un tel système sont :

$$\begin{array}{lll} r_1(u_1,u_2,\lambda) = & 2u_1 - u_2 + 100\frac{u_1}{1+u_1+u_1^2} - \lambda & = 0 \\ r_2(u_1,u_2,\lambda) = & 2u_2 - u_1 + 100\frac{u_2}{1+u_2+u_2^2} - (\lambda+\mu) & = 0 \end{array} \qquad [3]$$

Pour une valeur fixée du paramètre μ, on cherche l'évolution de u_1 et u_2 en fonction de λ. Il est établit dans [DOE 91] que pour $\mu = 0$, la branche de solution fondamentale présente deux bifurcations, reliées entre elles par une branche secondaire. Pour $\mu \neq 0$, on a cette fois des branches continues avec des quasi-bifurcations (variation brutale de courbure au voisinage des bifurcations).

2.2. *Mise sous forme quadratique : les opérateurs L0, L, et Q*

La première étape d'une MAN consiste à réécrire le système sous une forme quadratique qui se prête bien aux développements en séries. Par forme quadratique, on entend que les termes non linéaires doivent être polynômiaux et de degré deux au plus. Cette opération est réalisable sur la plupart des systèmes algébriques réguliers[3] en introduisant des variables et des équations supplémentaires adéquates dans le système (voir la section 5). Il y a en général de nombreuses façons d'arriver au résultat et on privilégie, en pratique, celle qui minimise le nombre de variables et d'équations supplémentaires.

Pour notre exemple, on introduit les quatre variables supplémentaires suivantes :

$$\begin{array}{lll} v_1 & = & u_1 + u_1^2 \\ v_2 & = & u_2 + u_2^2 \\ v_3 & = & \frac{1}{1+v_1} \\ v_4 & = & \frac{1}{1+v_2} \end{array} \qquad [4]$$

Elles sont reliées aux variables d'origine u_1,u_2 par des relations quadratiques, et permettent d'écrire [3] sous la forme quadratique suivante :

$$\begin{array}{llll} 0 & +2u_1 - u_2 - \lambda & +100u_1v_3 & = 0 \\ -\mu & +2u_2 - u_1 - \lambda & +100u_2v_4 & = 0 \\ 0 & +v_1 - u_1 & -u_1^2 & = 0 \\ 0 & +v_2 - u_2 & -u_2^2 & = 0 \\ -1 & +v_3 & +v_1v_3 & = 0 \\ -1 & +v_4 & +v_2v_4 & = 0 \end{array} \qquad [5]$$

3. Pour les systèmes non réguliers, il faut egalement procéder à une procédure de régularisation.

où l'on a rangé en colonnes les termes constants, linéaires et quadratiques. Ce dernier système de 6 équations est équivalent à [3].

Dans la suite, on souhaite ne pas faire jouer un rôle privilégié au paramètre λ. On définit pour cela le vecteur inconnu U comme :

$$U = [u_1, u_2, v_1, v_2, v_3, v_4, \lambda] \qquad [6]$$

et on adopte l'écriture formelle suivante pour le système :

$$R(U) = L0 + L(U) + Q(U,U) = 0 \quad R \in R^{N_{eq}} \; U \in R^{N_{eq}+1} \qquad [7]$$

où $L0$ est un vecteur constant, $L(.)$ un opérateur linéaire et $Q(.,*)$ un opérateur quadratique (linéaire par rapport à chacun des deux arguments . et $*$).

Nous précisons de suite l'implémentation de ces inconnues et équations sous Matlab. Le vecteur inconnu U sera simplement un vecteur ligne de taille $N_{eq}+1$, et les opérateurs $L0$,L et Q des fonctions Matlab prennant respectivement zéro, un ou deux arguments, et qui renvoient un vecteur colonne de taille $N_{eq}+1$. Pour notre exemple, les trois fichiers sources Matlab (fichiers .m) de ces fonctions sont :

```
function [L0] = L0

L0=zeros(6,1);
L0(1)=   0;
L0(2)=  -0.2;
L0(3)=   0;
L0(4)=   0;
L0(5)=  -1;
L0(6)=  -1;
```

```
function [L] = L(U)

L=zeros(6,1);
L(1)=2*U(1)- U(2) -U(7);
L(2)=2*U(2)- U(1) -U(7);
L(3)=U(3)-U(1);
L(4)=U(4)-U(2);
L(5)=U(5);
L(6)=U(6);
```

```
function [Q] = Q(U1,U2)

Q=zeros(6,1);
Q(1)=100*U1(1)*U2(5);
Q(2)=100*U1(2)*U2(6);
Q(3)=   -U1(1)*U2(1);
Q(4)=   -U1(2)*U2(2);
Q(5)=    U1(3)*U2(5);
Q(6)=    U1(4)*U2(6);
```

Dans un souci de généralité, nous allons dans la suite travailler essentiellement avec ces opérateurs $L0$,L et Q, Il suffira de changer ces fonctions $L0$,L et Q pour traiter d'autres exemples. La fonction Matlab qui évalue le résidu [7] est simplement :

```
function [R]=R(U)

R=L0+L(U)+Q(U,U);
```

2.3. *Calcul de la matrice jacobienne*

Pour un système quadratique de la forme [7], la matrice jacobienne dépend linéairement de U. Elle s'écrit à l'aide des opérateurs L et Q sous la forme :

$$\frac{\partial R}{\partial U} = L(.) + Q(U,.) + Q(.,U) \qquad [8]$$

Cette matrice à N_{eq} lignes et $N_{eq}+1$ colonnes sera notée $Lt_{(U)}$. En pratique, on la détermine par colonne de la facon suivante : si $e_j = [0,0,,,,1,0,0]$ est le vecteur canonique ayant 1 pour j^{ieme} composante, alors la colonne j de la matrice jacobienne vaut $\frac{\partial R}{\partial U} e_j = L(e_j) + Q(U,e_j) + Q(e_j,U)$. La fonction Matlab qui calcule cette matrice jacobienne est :

```
function [jacob] = jacob(U)

Neq=length(U)-1;
jacob=zeros(Neq,Neq+1);
Id=eye(Neq+1);    % matrice identite
for j=1:Neq+1
 Uj=Id(:,j);
 jacob(:,j) = L(Uj) + Q(U,Uj) + Q(Uj,U);
end
```

2.4. *Vecteur tangent à la branche, point solution orienté*

La fonction précédente peut servir, par exemple, à calculer le vecteur tangent T à la branche de solutions en un point U. Ce vecteur tangent, qui est solution du système sous déterminé $\frac{\partial R}{\partial U}T = 0$, est rendu unique en fixant arbitrairement sa dernière composante à $+1$. Les autres composantes sont alors solution d'un système linéaire carré où la matrice est composée des N_{eq} premières colonnes de $\frac{\partial R}{\partial U}$. Le fichier source Matlab est :

```
function T=tangente(U)

Neq=length(U)-1;
K=jacob(U);
T(Neq+1)=1;
T(1:Neq)=K(1:Neq,1:Neq)\(-K(1:Neq,Neq+1)) ;
```

Un point solution muni de son vecteur tangent sera appelé point solution orienté. Sous Matlab, on utilise un type structuré à deux champs pour ces objets

```
% point orienté Pa
Pa.U= [ 1 1 0 0 .... 0]
pa.T=tangente(Pa.U)
```

Remarque : On convient que le deuxième champ (vecteur tangent) ne sert qu'à indiquer le sens de continuation, et qu'il peut contenir une valeur approchée de la tangente.

2.5. *Calcul d'un tronçon de branche*

La branche de solution issue du point de départ U_0 est cherchée sous la forme :

$$U(a) = U_0 + aU_1 + a^2U_2 + \ldots + a^NU_N \qquad [9]$$

où a est un paramètre de chemin de type longueur d'arc généralisée défini par

$$a = U_1^t A(U - U0) \qquad [10]$$

Ici, A est une matrice diagonale permettant de sélectionner les composantes de U que l'utilisateur souhaite faire intervenir dans la longueur d'arc. On peut ainsi retenir au

choix : toutes les composantes de U ($A = Id$), les composantes d'origine (u_1,u_2,λ) sans les variables supplémentaires introduites pour rendre le problème quadratique (v_1,v_2,v_3,v_4), une seule composante de U [4], et éventuellement pondérer les composantes entre elles en mettant des coefficients sur la diagonale de A. Ces choix dépendent du problème à résoudre, et ils sont laissés à l'utilisateur. Dans le programme, la diagonale de la matrice A est un champ (LongArc) du type structuré para_man, qui contient les paramètres de la MAN (voir plus loin).

En reportant les séries [9] dans [7] et [10], et en ordonnant suivant les puissances de a, on obtient les systèmes d'équations linéaires qui déterminent tous les U_i. A l'ordre 1, il vient :

$$Lt_{(U_0)}(U_1) = 0 \qquad [11]$$

$$U_1^t A U_1 = 1 \qquad [12]$$

et à l'ordre p :

$$Lt_{(U_0)}(U_p) = -\sum_{k=1}^{p-1} Q(U_k, U_{p-k}) \qquad [13]$$

$$U_1^t A U_p = 0 \qquad [14]$$

Pour l'ordre 1, l'équation [11] entraîne que $U_1 = \alpha T$, où T est le vecteur tangent à la courbe en U_0. Le coefficient de proportionalité α est obtenu en reportant dans la seconde équation. On a $\alpha = \pm\frac{1}{\sqrt{T^t AT}}$. On choisit enfin le signe de α pour que le vecteur U_1 soit orienté dans le bon sens. L'ordre p est plus simple car les deux équations sont linéaires. On résout directement le système de taille $N_{eq}+1$ en rajoutant la ligne $U_1^t A$ à la matrice jacobienne.

Une fois les séries calculées, on en détermine le domaine de validité ; c'est-à-dire l'intervalle [0,a_m] du paramètre de chemin a sur lequel les séries satisfont le critère de tolérance $||R(U)|| < \epsilon_R$, où ϵ_R est une tolérance fixée par l'utilisateur. Ce calcul de a_m ne prend qu'une ligne de programme et ne réclame aucun calcul supplémentaire. On en rappelle ici le principe : pour évaluer la qualité de la série tronquée [9], il suffit de la reporter dans [7]. En ordonnant suivant les puissances de a, il vient

$$\begin{array}{rcl} R(U_0 + \ldots + a^N U_N) & = & R(U_0) + a(L_{t(U_0)}(U_1)) \\ & + & a^2(L_{t(U_0)}(U_2) + Q(U_1, U_1)) \\ & + & \ldots \\ & + & a^N(L_{t(U_0)}(U_N) + \sum_{k=1}^{N-1} Q(U_k, U_{N-k})) \\ & + & a^{N+1}(\sum_{k=1}^{N} Q(U_k, U_{N+1-k})) \\ & + & \ldots \\ & + & a^{2N} Q(U_N, U_N). \end{array} \qquad [15]$$

4. Ce choix n'est pas judicieux si la branche présente des points limites par rapport à cette composante.

Le premier terme $R(U_0)$ est nul par définition car U_0 est solution. Les N termes suivants correspondent exactement aux systèmes linéaires résolus à chaque ordre et ils sont également nuls. Le premier terme non nul est $a^{N+1}(\sum_{k=1}^{N} Q(U_k, U_{N+1-k}))$ et l'expérience montre qu'il est prépondérant devant les termes suivants. Ainsi, une très bonne approximation du résidu est obtenue par $R(U_0 + \ldots + a^N U_N) \simeq a^{N+1} F_N^{nl}$, où $F_N^{nl} = \sum_{k=1}^{N} Q(U_k, U_{N+1-k})$ est le second membre à l'ordre $N+1$.

La norme de $R(U(a))$ étant une fonction croissante de a, on est assuré que le critère $||R(U(a))|| < \epsilon_R$ est satisfait si a reste inférieur à la valeur maximale a_m suivante :

$$a_m = \left(\frac{\epsilon_R}{||F_N^{nl}||}\right)^{\frac{1}{N+1}} \qquad [16]$$

Dans le programme, un tronçon de branche est un type structuré qui contient : un point de départ orienté, les termes de séries évaluées en ce point, la valeur de a_m, et un point de fin orienté. Ce dernier est calculé comme , et sa tangente orientée dans le sens du parcours par — .

La fonction troncon.m, qui permet de calculer un objet tronçon, prend comme argument un point de départ orienté et un type structuré qui contient les paramètres de la MAN (voir plus loin).

```
function tronc=troncon(point_oriente,para_man)   % Calcul d'un troncon de branche

% tronc.point_oriente_dep      : un point oriente de depart
% tronc.series(NEQ+1,NORDRE+1) : les termes de serie (ligne de polynome Matlab)
% tronc.a_max                  : domaine de validite des series
% tronc.point_oriente_fin      : un point oriente de fin


Neq=length(point_oriente.U)-1; % nombre d'equations
Nordre=para_man.Nordre;        % ordre de troncature des series
U =zeros(Nordre,Neq+1);        % tableau local de stockage des series
K =zeros(Neq+1,Neq+1);         % matrice jacobienne augmentee
FP=zeros(Neq+1,1);             % second membre

tronc.point_oriente_dep=point_oriente;  % le point de départ est stocké dans l'objet

% ordre 1
%---------
   K(1:Neq,:)= jacob(point_oriente.U);
   T(Neq+1)=1;
   T(1:Neq)=K(1:Neq,1:Neq)\(-K(1:Neq,Neq+1)) ;
   alpha=1/sqrt(dot(T',para_man.LongArc.*T));
   U(1,:)=alpha*T;
   if dot(U(1,:),point_oriente.T) < 0  % orientation de U(1) dans le sens de point_oriente.T
       U(1,:)=-U(1,:);
   end
   FP(1:Neq) = -Q(U(1,:),U(1,:));    % preparation du second membre pour l'ordre 2

% les ordres suivants
%--------------------
   K(Neq+1,:)=para_man.LongArc.*U(1,:);
   for iordre=2:Nordre
       U(iordre,:)=(K\FP)';
       FP=zeros(Neq+1,1);             %preparation du second membre pour l'ordre p+1
       for ir=1:iordre
        FP(1:Neq)=FP(1:Neq) - Q(U(ir,:),U(iordre+1-ir,:));
       end
```

```
      norme_FNL=norm(FP(1:Neq));
  end


% stockage de UN, UN-1, ... , U1 puis U0 dans series
%-----------------------------------------------------
  tronc.series=zeros(Neq+1,Nordre+1);
  for i=1:Nordre
   tronc.series(:,i)=U(Nordre+1-i,:)';
  end
  tronc.series(:,Nordre+1)=point_oriente.U';


% calcul du domaine de validite des series
%----------------------------------------
  tronc.a_max=(para_man.Tolerance/norme_FNL)^(1/(Nordre+1)) ;


% On forme le point de sortie et sa tangente
%--------------------------------------------
  tronc.point_oriente_fin.U= point_oriente.U + ( tronc.a_max.^[1:Nordre])*U;
  for i=1:Nordre
     D(i)=i*(tronc.a_max)^(i-1);
  end
  tronc.point_oriente_fin.T= D*U;
```

2.6. *Continuation d'une branche*

Pour effectuer la continuation d'une branche, il suffit d'enchaîner le calcul de tronçons successifs, en reprenant le point orienté de fin d'un tronçon comme point de départ du tronçon suivant. La longueur des tronçons étant déterminée automatiquement par [16], l'utilisateur ne peut jouer que sur le nombre de tronçons pour aller plus ou moins loin sur la courbe.

Le programme qui effectue la continuation de notre exemple est donné ci-après. La branche est matérialisée comme un tableau (un vecteur) de tronçons. Toutes les fonctions ont été présentées, sauf corrige.m qui fait l'objet du paragraphe suivant et plot_courbes.m qui sert à tracer les courbes (données à la suite).

```
clear all
para_man.Nordre   = 20;                % ordre des series
para_man.Tolerance= 1e-04;             % tolerance sur le residu
para_man.LongArc  =[1 1 0 0 0 0 0];    % definition de la longueur d'arc

% on definit un premier point de depart approche, on le corrige et on verifie la qualité
%-----------------------------------------------------------------------------------------
U_ap=[0  0  0  0  1  1  0];
U0=corrige(U_ap,para_man);
norm(R(U0))

% a partir de U0, on forme un premier troncon de la branche 'fondam'
%-------------------------------------------------------------------
point_oriente.U=U0;
point_oriente.T=tangente(U0);
fondam(1)=troncon(point_oriente,para_man);

% puis on continue la branche en faisant d'autres troncons
%---------------------------------------------------------
for i=2:25
```

```
    fondam(i)=troncon(fondam(i-1).point_oriente_fin,para_man);
end

% on affiche la branche
%----------------------
plot_courbes(fondam,7,1,'b','r')
```

Les résultats de continuation pour $\mu = 0.1$ sont présentés sur les figures 2 et 3. Les cercles indiquent les débuts et les fins de pas. Après les deux premiers pas de continuation, où les concentrations u_1 et u_2 sont à peu près égales, les branches relatives à u_1 et u_2 se séparent au voisinage d'une quasi-bifurcation.

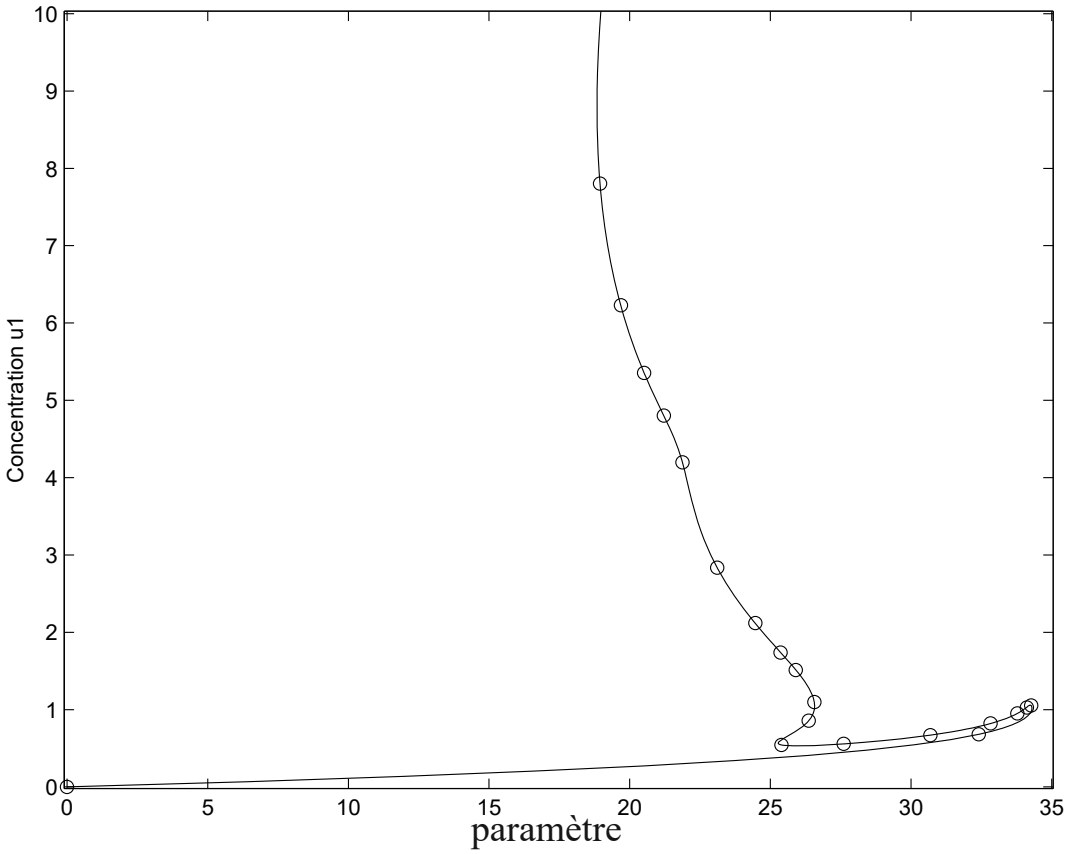

Figure 2. *Concentration en fonction de pour un paramètre perturbateur*

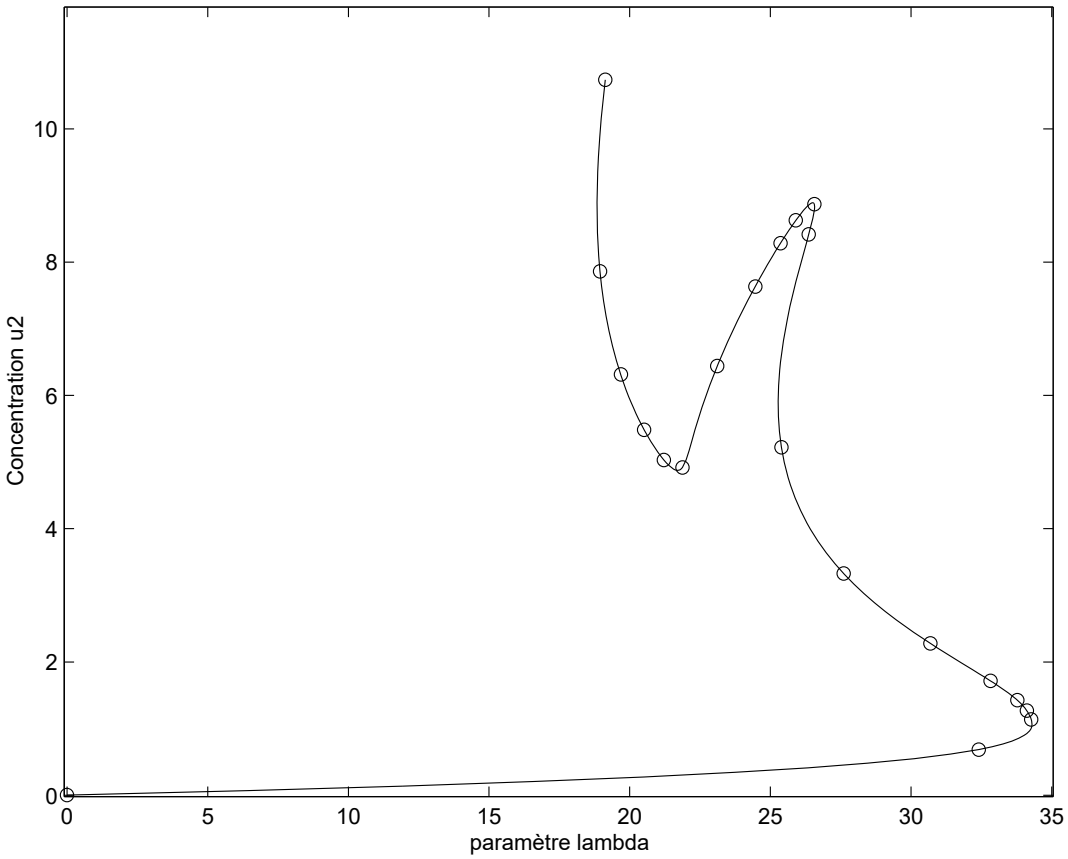

Figure 3. *Concentration u_2 en fonction de λ pour un paramètre perturbateur $\mu = 0.1$*

3. Détermination d'un premier point solution par homotopie

Avant de faire de la continuation, il faut d'abord déterminer un premier point solution sur la branche. Pour certains systèmes, on a parfois des solutions exactes pour des valeurs particulières des paramètres. Si ce n'est pas le cas, il faut avoir recours à une procédure numérique, telle qu'une correction de Newton par exemple, pour calculer ce premier point. Ici on va utiliser la technique d'homotopie proposée dans [MAL 00]. L'idée est de réaliser la continuation du paramètre d'homotopie sur l'intervalle [0,1] par une MAN.

Soit U_{ap} un point qui se trouve au voisinage de la courbe et dont le résidu $R(U_{ap}) = R_{ap}$ n'est pas trop grand[5]. On cherche une correction ΔU permettant de revenir sur la courbe, c'est-à-dire telle que :

$$R(U_{ap} + \Delta U) = Lt_{(U_{ap})}(\Delta U) + Q(\Delta U, \Delta U) + R_{ap} = 0 \qquad [17]$$

La correction ΔU ainsi définie n'est pas unique (ΔU est de taille $N + 1$) et il faut ajouter une condition supplémentaire qui précise la direction du retour. On peut par exemple fixer une composante de ΔU à zéro, ce qui revient à faire un retour parallèlement à l'un des axes. On peut aussi imposer que ΔU soit orthogonale à un vecteur donné. Par exemple, si U_{ap} est proche de la courbe, le vecteur U_t défini par

5. Un tel point est en général assez facile à construire.

$Lt_{(U_{ap})}U_t = 0$ sera localement parallèle à la courbe et il est raisonnable d'imposer la perpendicularité à ce vecteur (voir figure 4). Nous adoptons cette approche ici.

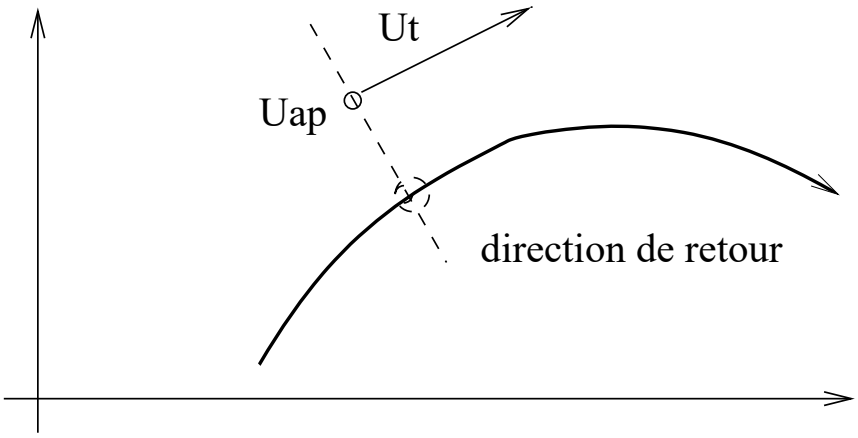

Figure 4. *Retour sur la branche perpendiculairement au vecteur U_t*

On introduit un paramètre d'homotopie ϵ dans le système de telle façon que pour $\epsilon = 0$ on ait la solution triviale $\Delta U = 0$, et pour $\epsilon = 1$ la solution cherchée.

$$\begin{array}{c} Lt_{(U_{ap})}(\Delta U) + Q(\Delta U, \Delta U) + \epsilon R_{ap} = 0 \\ U_t^t.\Delta U = 0 \end{array} \qquad [18]$$

On cherche ΔU sous la forme d'une série en ϵ : $\Delta U = \epsilon \Delta U_1 + \epsilon^2 \Delta U_2 + \ldots$ ce qui conduit à résoudre :

$$\begin{array}{l} \textit{Ordre 1} \left(\begin{array}{l} Lt_{(U_{ap})}(\Delta U_1) = -R_{ap} \\ U_t^t.\Delta U_1 = 0 \end{array} \right. \\ \textit{Ordre p} \left(\begin{array}{l} Lt_{(U_{ap})}(\Delta U_p) = -\sum_{k=1}^{p-1} Q(\Delta U_k, \Delta U_{p-k}) \\ U_t^t.\Delta U_p = 0 \end{array} \right. \end{array} \qquad [19]$$

Pour que le point $U_{ap} + \Delta U(\epsilon = 1)$ soit bien sur la courbe, il faut que le rayon de convergence des séries soit plus grand que un. Ceci sera le cas si le point U_{ap} n'est pas trop éloigné de la courbe[6].

```
function U_cor=corrige(U_ap,para_man)  % correction de U_ap par homotopie

Neq=length(U_ap)-1;                    % nombre d'equations
U =zeros(para_man.Nordre,Neq+1);       % stockages des series
K =zeros(Neq+1,Neq+1);                 % matrice jacobienne augmentee
FP=zeros(Neq+1,1);                     % second membre

 % ordre 1
 K(1:Neq,:)= jacob(U_ap);
 K(Neq+1,:)= tangente(U_ap);
 FP(1:Neq) = -R(U_ap);
 U(1,:)= (K\FP)';
 FP(1:Neq) = -Q(U(1,:),U(1,:));  % preparation du second membre pour l'ordre 2
```

6. Dans le cas contraire, on peut envisager de faire de la continuation par rapport à ϵ, c'est-à-dire de revenir en plusieurs pas.

```
% les ordres suivants
for iordre=2:para_man.Nordre
   U(iordre,:)=(K\FP)';
   FP=zeros(Neq+1,1);           % preparation du second membre pour l'ordre p+1
   for ir=1:iordre
     FP(1:Neq)=FP(1:Neq) - Q(U(ir,:),U(iordre+1-ir,:));
   end
end

% correction du point de depart
U_cor =U_ap +sum(U);
```

4. Comment effectuer un saut d'une branche à une autre

La continuation par MAN présentée à la section 2 possède la propriéte remarquable de toujours suivre la même branche sans faire de saut, et cela même lorsque la branche présente un changement très brutal de courbure [BAG 03]. C'est un avantage très appréciable pour la continuation, mais cela veut aussi dire qu'il faut une procédure spéciale pour qu'un utilisateur puisse effectuer volontairement un changement de branche.

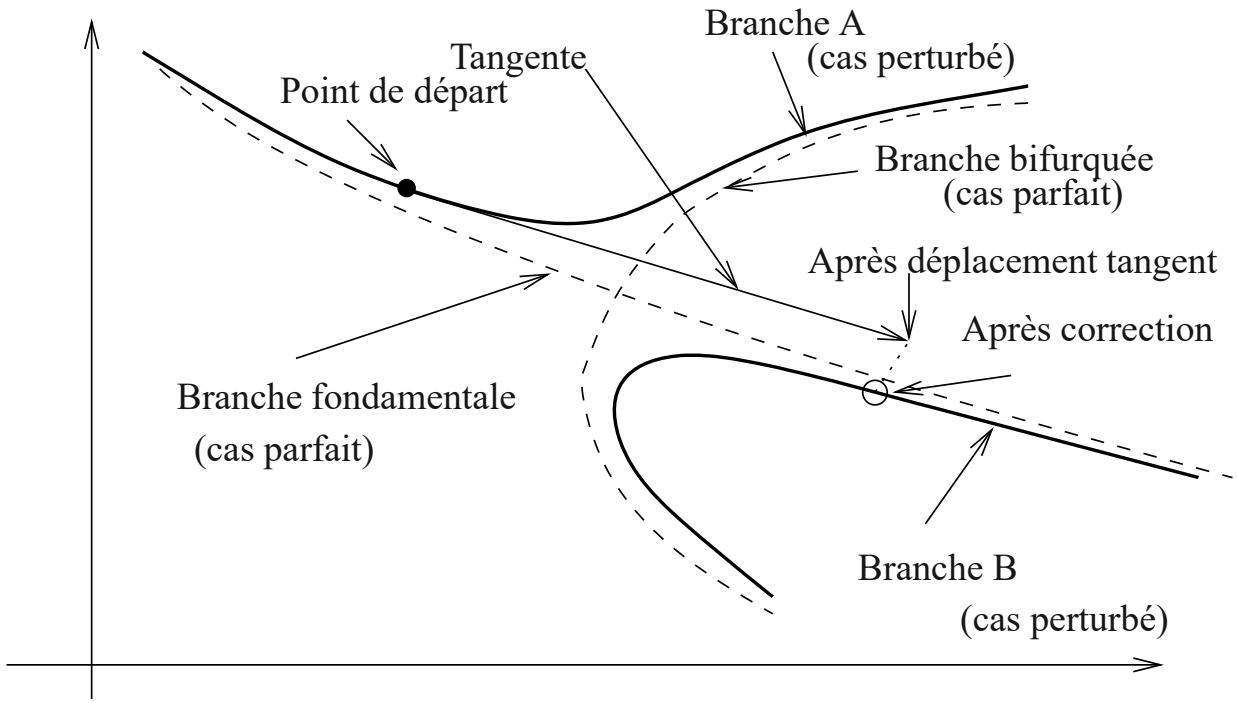

Figure 5. *Gestion d'un saut de branche par dessus une quasi-bifurcation*

Les fonctions déja présentées sont suffisantes pour effectuer un tel saut. Prenons par exemple le cas de la quasi-bifurcation (bifurcation perturbée) présentée à la figure 5. Dans cette situation, l'algorithme de continuation MAN suivra automatiquement la branche continue notée A. Pour passer sur l'autre branche [7] continue, il faut choisir un point de départ un peu en avant de la bifurcation, se déplacer le long de la tangente

7. Dans le but de continuer à suivre la branche fondamentale en définitive.

pour franchir la bifurcation, puis corriger selon la méthode de la section 3 pour revenir sur la branche B. Cette procédure de saut, qui nécessite toutefois un certain doigté, nous a permis de compléter le diagramme u_1, λ de notre exemple (voir figure 6).

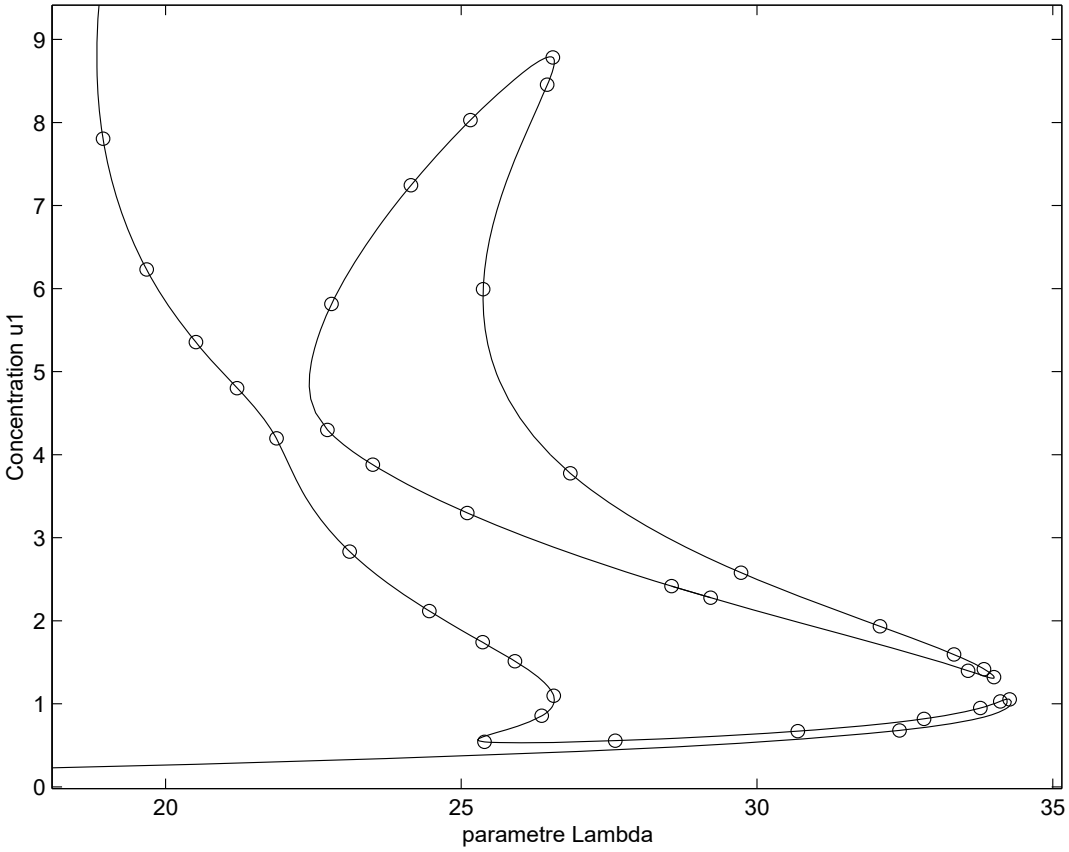

Figure 6. *Concentration en fonction de pour un paramètre perturbateur* $= 0\ 1$

5. Retour sur l'écriture quadratique des équations

Le programme de continuation présenté ici nécessite que les équations du système de départ [3] soient réécrites sous la forme quadratique [5], qui permet de programmer directement les fonctions L0, L et Q. Cette réécriture doit être réalisée par l'utilisateur qui est en droit de se poser la question suivante : « peut on mettre n'importe quel système de départ sous la forme quadratique [7] ? ». La réponse est certainement négative car on pourra très probablement toujours construire des contre-exemples, mais ce n'est pas une limitation forte : les techniques de transformation et de régularisation présentées dans [POT 97] permettent en définitive de rendre quadratique de nombreux problèmes de mécanique et de physique[8].

Prenons par exemple les systèmes utilisés par Seydel pour illustrer son livre sur la continuation et les bifurcations [SEY 94]. Ils sont issus de modèles de mécanique, de milieux réactifs, d'électronique, de dynamique des populations... Une bonne partie

8. Dans le cas où des régularisations sont introduites, il n'y a plus bien sur une équivalence stricte entre le système de départ et le système régularisé.

de ces systèmes présente des non-linéarités polynômiales de bas degré, des fractions rationnelles ou des racines carrées. Dans ce cas la mise sous forme quadratique est assez facile dès lors que l'on introduit le carré ou l'inverse de certaines inconnues, comme cela a été fait sur notre exemple. (Un autre exemple de ce type est encore présenté en section 5.1). La plupart des autres systèmes du livre de Seydel présentent des équations où les inconnues interviennent également dans des fonctions puissances (non entières), des exponentielles ou des fonctions trigonométriques. La mise sous forme quadratique est encore possible, mais en remplacant certaines équations par leur dérivées, comme cela est montré à la section 5.2. Quelques aménagements doivent être alors apportés au programme pour traiter ces cas.

5.1. *Vibrations non linéaires d'un système à un ddl*

Les oscillations forcées non linéaires d'un système masse ressort à un degré de liberté sont gouvernées par l'équation standard (Duffing) :

$$\ddot{u} + 2\mu\dot{u} + u + u^3 = f cos(\Omega t) \qquad [20]$$

On en cherche des solutions périodiques approchées sous la forme d'une série de Fourier tronquée $u(t) = u_c cos(\Omega t) + u_s sin(\Omega t)$. En reportant dans l'équation et en négligeant les harmoniques supérieures (technique de l'équilibrage harmonique), on obtient le système algébrique [21] où les inconnues sont u_c,u_s et le paramètre Ω . La force f et le facteur d'amortissement μ sont des paramètres fixés.

$$\begin{array}{l} (1-\Omega^2)u_c + 2\mu\Omega u_s + \frac{3}{4}u_c^3 + \frac{3}{4}u_c u_s^2 = f \\ (1-\Omega^2)u_s - 2\mu\Omega u_c + \frac{3}{4}u_s^3 + \frac{3}{4}u_s u_c^2 = 0 \end{array} \qquad [21]$$

On introduit les nouvelles variables $v_c = u_c^2$, $v_s = u_s^2$ et $\lambda = \Omega^2$ pour obtenir finalement :

$$\begin{array}{llll} -f & +u_c & -\lambda u_c + 2\mu\Omega u_s + \frac{3}{4}u_c v_c + \frac{3}{4}u_c v_s & = 0 \\ 0 & +u_s & -\lambda u_s - 2\mu\Omega u_c + \frac{3}{4}u_s v_s + \frac{3}{4}u_s v_c & = 0 \\ 0 & +v_c & -u_c u_c & = 0 \\ 0 & +v_s & -u_s u_s & = 0 \\ 0 & +\lambda & -\Omega\Omega & = 0 \end{array} \qquad [22]$$

Un exemple de continuation est présenté sur la figure suivante.

5.2. *Un modèle de circuit électronique*

Dans [SEY 94], l'auteur analyse un circuit électronique composé de résistances, de diodes et d'un amplificateur. Si I est l'intensité du courant et U la différence de potentiel, les résistances sont modélisées par $I = \frac{U}{R}$, les diodes par $I_d(U) = \alpha(e^{25U} - 1)$ et l'amplificateur par $U_a(U) = \beta \arctan(1962U)$. Nous n'avons pas l'intention de décrire le circuit complet mais simplement de montrer comment rendre quadratique

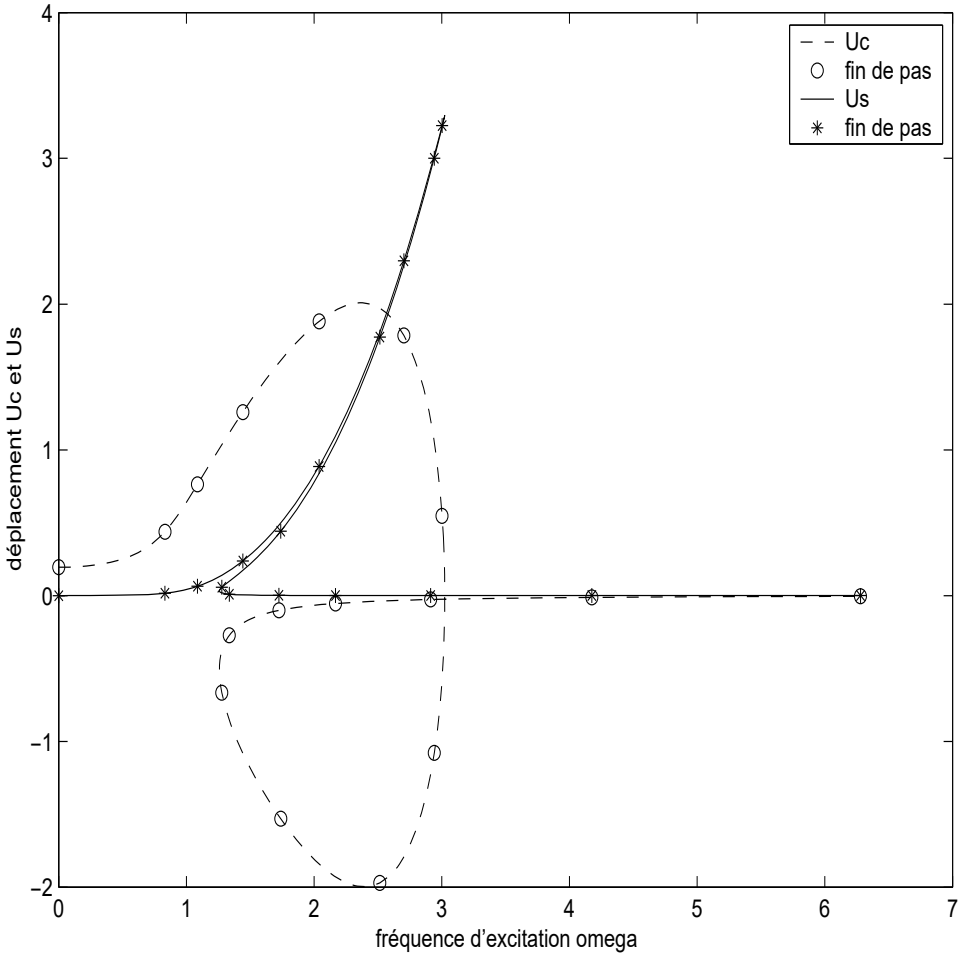

Figure 7. *Evolution des composantes u_c et u_s du déplacement en fonction de la fréquence d'excitation Ω*

une équation du système (loi de Kirchhoff) qui ferait intervenir ces trois composants. Prenons par exemple :

$$\frac{(U_6 - U_2)}{R_1} + \alpha(e^{25U_6} - 1) + \frac{\beta \arctan(1962(U_3 - U_4)) - U_6}{R_2} = 0 \qquad [23]$$

où $U_1, \ldots, U_6$ sont les potentiels aux nœuds du circuit qui concernent ces composants. On commence par introduire les nouvelles variables V_1,V_2,V_3 :

$$\begin{aligned} V_1 &= e^{25U_6} \\ V_2 &= 1962(U_3 - U_4) \\ tan(V_3) &= V_2 \end{aligned} \qquad [24]$$

qui permettent de réécrire [23] sous une forme linéaire. On remplace ensuite la première et la dernière équation de [24] par leurs dérivées par rapport au paramètre a, car elles se prêtent bien ainsi à une mise sous forme quadratique :

$$\begin{aligned} \frac{dV_1}{da} &= 25e^{25U_6}\frac{dU_6}{da} \\ (1 + tan^2(V_3))\frac{dV_3}{da} &= \frac{dV_2}{da} \end{aligned} \qquad [25]$$

On introduit enfin $V_4 = V_2^2$, pour obtenir le système quadratique final :

$$\begin{array}{c} \frac{(U_6-U_2)}{R_1} + \alpha(V_1 - 1) + \frac{\beta V_3 - U_6}{R_2} = 0 \\ \frac{dV_1}{da} = 25 V_1 \frac{dU_6}{da} \\ V_2 = 1962(U_3 - U_4) \\ (1+V_4)\frac{dV_3}{da} = \frac{dV_2}{da} \\ V_4 = V_2^2 \end{array} \qquad [26]$$

Le calcul des séries n'est pas plus compliqué pour ce système que pour les précédents. Il faut toutefois adapter le programme précedent car l'algèbre est légèrement différente à cause des termes dérivées. En effet, si $U = U_0 + aU_1 + a^2U_2 + \ldots$ on a alors $\frac{dU}{da} = U_1 + a(2U_2) + a^2(3U_3) + \ldots$. Ainsi, le système d'équations linéaires qui gouverne U_k est constitué des équations [26] écrites à l'ordre k pour la première, la troisième et la cinquième, et à l'ordre $k-1$ pour les deux autres. Une extention du programme permettant de traiter systématiquement des systèmes de type [23] sera développée ultérieurment.

6. Conclusions

Dans ce papier, nous avons rappelé les concepts qui sous-tendent les Méthodes Asymptotiques Numériques, et détaillé un programme Matlab qui effectue la continuation des systèmes algébriques mis sous la forme canonique [7].

Les fonctions Matlab présentées ici ont surtout pour objet de montrer que la mise en œuvre est simple et facilement modifiable par un utilisateur en fonction de ses besoins. Cette première réalisation pourra constituer le point de départ d'un véritable outil logiciel de continuation, basé sur une programmation orientée objet afin d'assurer la pérénnité des développements. De nouvelles fonctionnalités telles que la détection des bifurcations (stationnaire, Hopf) ou le suivi de branches de points singuliers seront progressivement ajoutées.

Le calcul automatique et systématique de toutes les branches d'un système dans un domaine délimité est un défi scientifiquement intéressant. Cela peut d'ailleurs devenir un véritable casse-tête lorsque les branches sont nombreuses, proches les unes des autres, ou en forme d'anneau. Dans la pratique cependant, un tel travail est rarement utile car un utilisateur écarte très vite les solutions non physiques, les solutions sans intérêt, les solutions instables, et se concentre sur quelques branches seulement. Ce sont plutôt les aspects interactifs et graphiques qui priment. Il faut permettre à l'utilisateur de commander ses actions à la souris et de visualiser les effets en direct. Une interface graphique allant dans ce sens est en cours de développement.

7. Bibliographie

[ALL 90] ALLGOWER K., GEORG K., *Numerical continuation methods, an introduction*, vol. 13 of Springer series in Computational Mathematics, Springer-Verlag, New-york, 1990.

[AZR 93] AZRAR L., COCHELIN B., DAMIL N., POTIER-FERRY M., « An asymptotic numerical method to compute the post-buckling behavior of elastic plates and shells », *International Journal for Numerical Methods in Engineering*, vol. 36, 1993, p. 1251-1277.

[BAG 03] BAGUET S., COCHELIN B., « On the behaviour of the ANM in the presence of bifurcations », *Communications in Numerical Methods in Engineering*, vol. 19, 2003, p. 459-471.

[COC 94] COCHELIN B., DAMIL N., POTIER-FERRY M., « The Asymptotic-Numerical-Method : an efficient perturbation technique for nonlinear structural mechanics », *Revue Européenne des Éléments Finis*, vol. 3, 1994, p. 281-297.

[CRI 97] CRISFIELD M., *Non-linear finite element analysis of solids and structures*, vol. 2, Wiley, New York, 1997.

[DAM 90] DAMIL N., POTIER-FERRY M., « A new method to compute perturbed bifurcations : application to the buckling of imperfect elastic structures », *International Journal of Engineering Sciences - N9*, vol. 28, 1990, p. 943-957.

[DOE 91] DOEDEL E., KELLER H., KERNEVEZ J., « Numerical analysis and control of bifurcation problems (i) bifurcation in finite dimensions », *International Journal of Bifurcation and Chaos*, vol. 1, 1991, p. 493-520.

[KEL 87] KELLER H., « Lectures on numerical methods in bifurcation problems », *Notes by Nandakumaran and Ramaswamy*, Bangalore, 1987, Springer-Verlag.

[MAL 00] MALLIL E., LAHMAM H., DAMIL N., POTIER-FERRY M., « An iterative process based on homotopy and perturbation technique », *Computer Methods in Applied Mechanics and Engineering*, vol. 190, 2000, p. 1845-1858.

[POT 97] POTIER-FERRY M., DAMIL N., BRAIKAT B., DESCAMPS J., CADOU J., CAO H., ELHAGEHUSSEIN A., « Traitement des fortes non-linéarités par la méthode asymptotique numérique », *Compte Rendu de l'Académie des Sciences*, vol. 314, 1997, p. 171-177, Serie II b.

[SEY 94] SEYDEL R., *Practical bifurcation and stability analysis, from equilibrium to chaos*, Springer-Verlag, New York, 1994.