Projet SIM202

Modèle numérique pour la détection de mines

Puwei Liao, M1 Orsay
Ebénezer Ogounkola
Mario Gervais, M1 Orsay

11 décembre 2021

Chargé de TP
Vincent Rouvreau
INRIA
École Polytechnique, 91128 Palaiseau
https://inria.fr/fr

Table des matières

1 Introduction

1.1 Présentation du problème

Le but de ce projet est de simuler la propagation d'une onde elastodynamique dans le sol, ce qui constituerait un travail préliminaire au développement d'un détecteur d'objet enfouis.
La modélisation du problème est la suivante : on dispose d'un milieu comme sur la Figure 1. Ce milieu est constitué de roche, de terre, et d'objets à détecter, par exemple, des mines en acier. Á la surface, on applique une contrainte mécanique, et on simule la propagation de cette contrainte dans le milieu.

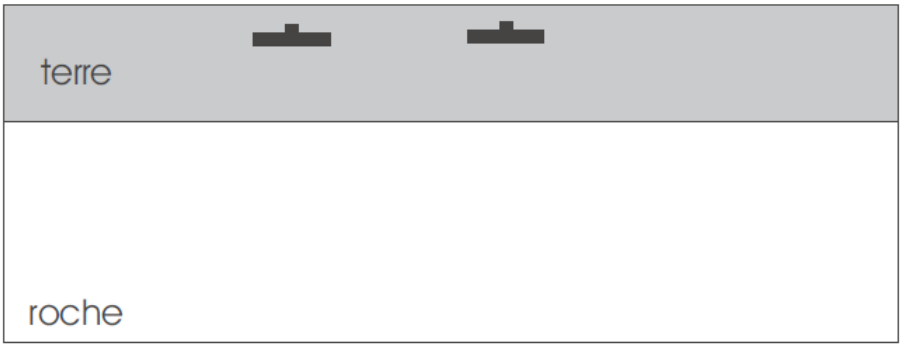


FIGURE 1 – Milieu étudié dans lequel l'onde va se propager

La chaque milieu $s = r, t, m$, les équations de l'élastodynamique s'écrivent comme suit :

$$\rho_s \frac{\partial^2 \vec{u}_s}{\partial t^2} - \vec{\nabla} \cdot (\sigma(\vec{u}_s)) = 0 \tag{1}$$

Pour les conditions aux limites, on considère :

1. la surface : on a ici une contrainte appliquée à la terre qui est la source de l'onde que nous allons simuler
2. les côtés et le fond : sur cette partie de la frontière (qui est artificielle, car nous ne pouvons simuler un domaine infini), on considère une condition "absorbante du premier ordre"

Les formulations mathématiques de ces conditions aux limites se trouvent dans le document présent en annexe.

1.2 Discrétisation du problème

En introduisant sa formulation variationnelle (sous forme intégrale), on montre que c'est un problème bien posé, dont la solution existe, est unique et régulière. Nous pouvons alors discrétisé ce problème pour le résoudre numériquement.
Supposons que l'on dispose d'un maillage de taille N dont les fonctions de base soient les ω_i, fonction de Lagrange P1. Pour rappel, ω_I vaut 1 sur S_I, 0 sur les autres sommets, et est un polynôme degré 1.
Introduisons alors les fonctions

$$\vec{\omega_I}^1 = \begin{pmatrix} \omega_I \\ 0 \end{pmatrix} \quad \vec{\omega_I}^2 = \begin{pmatrix} 0 \\ \omega_I \end{pmatrix}$$

et

$$V_h =< \vec{\omega_1}^1, ..., \vec{\omega_N}^1, \vec{\omega_1}^2, ..., \vec{\omega_N}^2 >$$

Si $u_h \in V_h$, les N premières composantes représentent le champ de déplacement selon la direction x, et les N suivantes représentent le champ selon la direction y. Le problème à résoudre est le suivant :

Trouver $u_h \in V_h$ telle que $\forall v \in V_h$:

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}\int_\Omega \rho\vec{u}_h \cdot \vec{v} + \int_\Omega \lambda\vec{\nabla}\cdot\vec{u}_h\vec{\nabla}\cdot\vec{v} + + \int_\Omega 2\mu\epsilon(\vec{u}_h)\epsilon(\vec{v}) - \frac{\mathrm{d}}{\mathrm{d}t}\int_\Sigma \rho\mathbb{A}\vec{u}_h\cdot\vec{v} = \int_\Gamma \vec{f}\cdot v \qquad (2)$$

Ici, les coefficients λ, μ, ρ (qui sont différents selon le milieu), la matrice $\mathbb{A}$ et la fonction $\vec{f}$ sont les données du problème.

Si on renomme $\vec{u_h}$ en $\vec{U_h}$, et décomposer ce vecteur sur la base des $\vec{\omega_I}^i$, $i = 1, 2$ et $I = 1, .., N$. Ainsi, les N *premières coordonnées de* $\vec{U}$ *seront les composantes en* x *du champ de déplacement, tandis que les* N *suivantes seront les composantes en* y.

On introduit aussi les matrices éléments finis (cf Annexe) :

1. $\mathbb{K}$, matrice de rigidité, $(\mathbb{K}_{IJ})_{ij} = \int_\Omega \lambda\ div\vec{\omega}_I^i\ div\vec{\omega}_J^j \mathrm{d}\Omega + \int_\Omega 2\mu\ \epsilon(\vec{\omega}_I^i)\ \epsilon(\vec{\omega}_J^j)\ \mathrm{d}\Omega$
2. M, matrice de masse, $(\mathbb{M}_{IJ})_{ij}$
3. $\mathbb{T}$, matrice de masse surfacique, $(\mathbb{T}_{IJ})_{ij} = \int_\Sigma \rho\ \mathbb{A}\ \vec{\omega}_I^i\cdot\vec{\omega}_J^j\ \mathrm{d}\Omega$

   Notons que Σ représente l'union des côtés ainsi que du fond du milieu, mais *pas* du "haut" du mileu. L'intégrale sur cette partie du bord sera utilisée plus tard, dans la construction du vecteur $\vec{F}$.
4. $\mathbb{D}$ matrice de masse condensée. $\mathbb{D}_{ll} = \sum_{p=1,2N} \mathbb{M}_{lp}$

La formulation variationnelle précédente s'écrit alors :

$$\mathbb{M}\frac{\mathrm{d}^2}{\mathrm{d}t^2}\vec{U}(t) + \mathbb{K}\vec{U}(t) - \frac{\mathrm{d}}{\mathrm{d}t}\vec{U}(t) = \vec{F}$$

(3)

Il s'agit d'un système différentiel que l'on peut résoudre en appliquant un schéma saute-mouton.
Après quelques calculs, nous arrivons au schéma numérique suivant :

$$\vec{U}_{k+1} = 2\vec{U}_k - \vec{U}_{k-1} + \Delta t\mathbb{D}^{-1}\left(\vec{F}_k + \mathbb{K}\vec{U}_k + \mathbb{T}\frac{\vec{U}_k - \vec{U}_{k-1}}{\Delta t}\right) \qquad (4)$$

où $\vec{U_k}$ représente le vecteur $\vec{U}$ aux différents instants.

Le vecteur $\vec{F}$ représente le vecteur source : c'est l'excitation que l'on fournit à la surface pour induire le champ déplacement que l'on simule. Mathématiquement, il se calcule comme suit :

$$\vec{F} = \left(\begin{array}{c|c} \mathbb{S} & 0 \\ \hline 0 & \mathbb{S} \end{array}\right) \left(\begin{array}{c} \vec{F_1} \\ \hline \vec{F_2} \end{array}\right)$$

où

— $\mathbb{S}_{I,J} = \int_{\Sigma_S} \omega_I \omega_J$, l'intégrale est prise sur la surface du domaine Ω, ce qui est normal puisque l'excitation est à la surface,

— $(\vec{F_1})_I = f_1(x_I, t)$, où x_I désigne l'abscisse du sommet I,

— $(\vec{F_2})_I = f_2(x_I, t)$.

La stratégie que nous avons utilisée est la suivante :

— Développer les classes Vecteur, Matrice, la classe Maillage et ses classes associées,

— Écrire l'algorithme d'assemblage, qui va construire les matrices éléments finis,$\mathbb{M}$, $\mathbb{K}$, $\mathbb{T}$, $\mathbb{S}$.

— Écrire les coordonnées de $\vec{U}$ dans un fichier .txt, que nous lisons ensuite avec Matlab.

2 Développement des classes

Nous développons donc d'abord les classes Vecteur et Matrice, avant de détailler la construction de la classe Maillage.

2.1 Classes basiques

On construit une classe vecteur héritant de $std :: vector < double >$, en développant les opérateurs produit, division, addition et soustraction entre 2 vecteurs, et les opérateurs produit et division par un scalaire.

On construit en ensuite la classe Matrice. Cette classe possède les opérations traditionnelles de la classe Matrice (accesseurs, somme, produit par un vecteur). On doit aussi développer le produit Matrice Vecteur, et le produit matriciel.
Notons quand même une forte limitation de notre classe Matrice. Dans notre situation, les matrices sont creuses (si i et j sont deux sommets du maillage, les matrices $\mathbb{M}$, T, et $\mathbb{K}$ ne seront non-nuls que si i et j sont situés dans un même triangle). Le problème est que notre classe matrice stocke *tous* les coefficients, ce qui est très loin d'être optimal :

```
class Matrice
{
    private:
        int dim_l,dim_c;
        vector<vector <double> > mat;
};
```

Pour remédier à ce problème, nous avions pensé à utiliser le stockage profil, mais nous sommes tombés sur un problème dans l'accès en écriture.
Enfin, nous avons développé un fonction appelée construction, de prototype :

```
Matrice construction(Matrice M1_1, Matrice M1_2,Matrice M2_1, Matrice M2_2);
```

qui permet, à partir de 4 matrices extraites, $M_{1,1}, M_{1,2}, M_{2,1}, M_{2,2}$ de construire la matrice complète :

$$\mathbb{M} = \left(\begin{array}{c|c} M_{1,1} & M_{1,2} \\ \hline M_{2,1} & M_{2,2} \end{array}\right)$$

Nous avons développé cette fonction car dans la procédure d'assemblage, qui constitue le coeur de l'algorithme, nous travaillons directement avec les matrice extraites.

2.2 Classe Maillage

Maintenant que nous avons nos classes basiques, nous pouvons passer à la classe Maillage. Pour rappel, c'est grâce à cette classe que nous allons calculer les matrices éléments finis, ce qui en fait la classe fondamentale du programme. Les données de la classe maillage sont les suivantes :

```
class Maillage
{
    public:
        vector<Point2D> Noeuds;      //vecteur de point : Noeuds[i] contient
                                     //la position du i-ème noeud

        vector<Numeros> Num_Tri;     //vecteur de triplet : Num_Tri[l] contient les numéros
                                     //des sommets compposant le triangle numéro l

        vector<int> Zone;            //Zone[l] contient la zone dans laquelle est située le
                                     //l-ième triangle(1 ->roche, 2 -> terre, 3 -> mines)

        vector<Arete> Surface;
        //vecteur de qintuplet : Surface[i] contient :
       //le premier et le second indice représentent le numéro des sommets de l'arrête,
        //le 3 ème est les numéros du sommets formant un triangle avec les 2 précédents,
        //le 4ème est l'indice de la surface :
        //0 pour la surface,1 pour les cotés, 2 pour le fond
        //le dernier est la zone dans laquelle se situe l'arête.
};
```

Nous devons donc écrire les classes associées à la classe Maillage :

- la classe Point2D, qui est une classe de doublets,
- la classe Numeros, où chaque objet est un triplet d'entiers,
- la classe Arete, où chaque objet est un quintuplet d'entiers

On suppose donc à présent ces clases développées.

Puisque nous générons le maillage avec gmsh, nous devons construire l'objet de la classe Maillage avec une fonction de lecture dans un fichier msh :

```
void lecture_msh(vector<Point2D> & list_noeuds, vector<Numeros> & list_num ,
vector<int> & ist_zone,vector<Arete> & list_aretes, vector<vector <int> > & M);
```

qui modifie

1. *list_noeuds* pour la transformer en le tableau $Noeuds$,
2. *list_num* $\rightarrow Num$,
3. *list*$_{\rightarrow}$*Zone*.

Pour cela, il est nécéssaire de tenir compte de la structure du fichier.txt renvoyé par gmsh. Voici un exemple de fichier que nous devons lire :

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
21
1 0 0 0
2 2 0 0
3 2 2 0
4 0 2 0
5 0.6 0.2 0
6 1.8 0.2 0
7 1.8 1.8 0
8 0.6 1.8 0
9 0.9999999999997388 0 0
10 2 0.9999999999997388 0
11 1.000000000004118 2 0
12 0 1.000000000004118 0
13 1.199999999998331 0.2 0
14 1.8 0.9999999999979183 0
15 1.200000000001388 1.8 0
16 0.6 1.00000000000206 0
17 0.3 0.5500000000015446 0
18 0.3 1.450000000001545 0
19 1.019999999999594 0.6275000000002851 0
20 1.208746355685375 1.185969387755244 0
21 1.432678050811661 0.6304956268216237 0
$EndNodes
$Elements
52
1 15 2 1 1 1
2 15 2 1 2 2
3 15 2 1 3 3
4 15 2 1 4 4
5 1 2 1 1 1 9
6 1 2 1 1 9 2
7 1 2 1 2 2 10
8 1 2 1 2 10 3
```

FIGURE 2 – Fichier .txt généré par gmsh pour un maillage à 21 noeuds

Concrètement, on procède la manière suivante :
Dans un premier temps, on lit les coordonnées des sommets et on les met dans le tableau *list_noeuds* :

```
for (int i = 0; i < nb_noeuds; i++) {
     fich>>a;          //num de noeuds//
     fich>>coord_x;
     fich>>coord_y;
     fich>>a;      //coord_z, inutile ici//
     p=Point2D(coord_x,coord_y);
     list_noeuds[i]=p;
}
```

Une fois que l'on a fini ceci, on passe aux "Elements" du fichier .msh.Cet ensemble d'objets contient les triangles, les segments (arêtes), et les sommets, et indique dans quelle zone du

maillage ils seront situés. Dans notre cas, on a 3 zones : la terre, la roche, et les mines. On rajoute les triangles dans le tableau *list_num*, et la zone dans *list_zone*.
Pour le tableau Surface, nous devons garder une trace des sommets reliés par un triangle : c'est ce que fait la matrice M, construite lors de l'appel à *lecture_msh* : en $M_{n_1,n_2} = 0$ si les deux sommets ne sont pas reliés par un triangle, et vaut n_3 sinon, où n_3 est le sommet formant le triangle.

```
Numeros N;
    Arete A;
    for (int i = 0; i < nb_ele; i++)
    {
         fich>>a;
         fich>>b;
         if(b==2){      //on traite le cas triangle//
             fich>>c;
             fich>>domaine;
             list_zone.push_back(domaine);
             fich>>c;
             fich>>n1;
             fich>>n2;
             fich>>n3;
             N=Numeros(n1,n2,n3);
             list_num.push_back(N);

             n1--;n2--;n3--;
             M[n1][n2]=n3;
             M[n1][n3]=n2;
             M[n2][n3]=n1;

             M[n2][n1]=n3;
             M[n3][n1]=n2;
             M[n3][n2]=n1;
         }
         else if(b==1)      //on traite le cas segment
         {
             fich>>c;
             fich>>domaine;
             fich>>c;
             fich>>n1;
             fich>>n2;
             A=Arete(n1,n2,-1,-1,domaine);
             list_aretes.push_back(A);
         }
         else{getline(fich,s);
         }
    }
```

Le constructeur de Maillage s'écrit donc comme suit :

```
Maillage::Maillage (){
    vector<Numeros> list_num;
    vector<int> list_zone ;
    vector<Point2D> list_noeuds;
    vector<Arete> list_aretes;
    vector <vector <int> > M;
    lecture_msh(list_noeuds,list_num,list_zone,list_aretes, M);

    Noeuds=list_noeuds;
    Num_Tri=list_num;
    Zone=list_zone;
    ...             //on modifie list_aretes grâce à la matrice M
    Surface=list_aretes;

}
```

3 Algorithme d'assemblage

À présent que nous disposons de notre classe Maillage, nous pouvons passer au coeur de l'algorithme : la construction des différentes matrices.

3.1 Matrice de masse et de rigidité $\mathbb{M}$ et $\mathbb{K}$

Pour les matrices $\mathbb{M}$ et $\mathbb{K}$, la construction est identique :

1. on parcourt les triangles du maillage,
2. on récupère les 3 sommets du triangle grâce au tableau Num_Tri,
3. on calcule la matrice élémentaire associée au triangle,
4. on ajoute à la bonne place les coefficients de la matrice élémentaire aux coefficients déjà présents dans la matrice assemblée.

Évidemment, nous devons faire cette procédure pour les 8 matrices $M_{1,1}, M1, 2, M_{2,1}, M_{2,2}$ (respectivement $\mathbb{K}$).

La différence entre $\mathbb{K}$ et $\mathbb{M}$ réside dans le calcul de la matrice élémentaire du triangle ; le reste est très similaire.

Voici le code complet de l'assemblage des matrices $M_{1,1}, M_{1,2}, M_{2,1}, M_{2,2}$.

```
void assembl_masse(Matrice& M1_1,Matrice& M1_2, Matrice& M2_1,Matrice& M2_2,
Maillage M, double* rho)
{
    int nb_tri = M.nb_tri();

    for(int l =1 ; l<= nb_tri ; l++)
    {
        Numeros T=M.num_tri(l);    //T est le i-ème triangle du maillage

        //les points i,j,k sont les points formant le triangle T_l
        Point2D i = Point2D( M.noeud(T(1)) );
        Point2D j = Point2D( M.noeud(T(2)));
        Point2D k = Point2D( M.noeud(T(3)));
```

```
        double x_i,x_j,x_k,y_i,y_j,y_k;     //on simplifie les notations
        x_i = i.Coor_x() ; x_j=j.Coor_x() ; x_k=k.Coor_x();
        y_i= i.Coor_y() ; y_j=j.Coor_y() ; y_k=k.Coor_y();

        //on met le rho correspondant à la zone dans laquelle se trouve le triangle
        double r=rho[M.zone(l)-1];

       //déterminant du changement de variables
        double D= abs((x_j-x_i)*(y_k-y_i) -(x_k-x_i)*(y_j-y_i));

           //matrice élementaire
        Matrice Mel=mat_masse_elem(D,r);

         //on met les coefficients à la bonne place dans la matrice
        M1_1(T(1),T(1)) +=Mel(1,1);
        M1_1(T(1),T(2)) += Mel(1,2);
        M1_1(T(1),T(3)) +=Mel(1,3);

        M1_1(T(2),T(1)) +=Mel(2,1);
        M1_1(T(2),T(2)) += Mel(2,2);
        M1_1(T(2),T(3)) +=Mel(2,3);

        M1_1(T(3),T(1)) +=Mel(3,1);
        M1_1(T(3),T(2)) += Mel(3,2);
        M1_1(T(3),T(3)) +=Mel(3,3);

    }

    M2_2 = M1_1;
}
```

Voici le calcul de la matrice de masse élémentaire de masse :

```
//matrice de masse elementaire
//On intègre les fonctions élémentaires sur le triangle,
//en transformant le triangle courant en triangle de base (0,0);(1,0);(0,1)
//D est le déterminnant du changement de variable
//rho est la masse volumique du milieu dans lequel on se trouve.


Matrice mat_masse_elem(double D,double rho)       //matrice de masse élementaire
{
    Matrice Mel = Matrice(3,3);

    Mel(1,1) =2;
    Mel(1,2)=1;
    Mel(1,3)= 1;
    Mel(2,1)=1;
    Mel(2,2)= 2;
```

```
    Mel(2,3) = 1;
    Mel(3,1)=1;
    Mel(3,2)= 1;
    Mel(3,3)=2;

    Mel *=rho*D/24;
    return Mel;
}
```

Remarque 1. *Notons que le calcul de* $\mathbb{K}$ *est un peu plus délicat, car :*

$$\mathbb{M} = \left(\begin{array}{c|c} M_{1,1} & 0 \\ \hline 0 & M_{2,2} = M_{1,1} \end{array}\right) \quad \textit{alors que} \quad \mathbb{K} = \left(\begin{array}{c|c} K_{1,1} & K_{1,2} \\ \hline K_{2,1} & K_{2,2} \end{array}\right)$$

3.2 Matrices surfaciques $\mathbb{T}$ et $\mathbb{S}$

Ici, le calcul de la matrice surfacique $\mathbb{T}$ est différent : il ne s'agit plus de calculer des intégrales volumiques, mais des intégrales surfaciques. Pour cela, l'algorithme s'écrit comme suit :

1. on parcourt les arêtes de la frontière du domaine, grâce au tableau $Surface$,
2. grâce au 4ème élément de $Surface[i]$, on test si l'arête est située sur le "fond" du domaine, sur un des bords, ou à la surface,
3. si on est dans un des deux premiers cas, on calcule la matrice élémentaire associée à cette arête.
4. on ajoute à la bonne place les coefficients de la matrice élémentaire aux coefficients déjà présents dans la matrice assemblée.

Pour la matrice $\mathbb{S}$, c'est *exactement* le même algorithme, mais l'étape 3 est changée en "si on est dans le dernier cas" : en effet, pour cette matrice, l'intégrale est sur la *surface* du domaine, alors que pour la matrice $\mathbb{T}$, l'intégrale est sur les *bords et sur le fond* de notre domaine.

Voici par exemple l'assemblage de la matrice $\mathbb{T}$, qui est légèrement plus compliquée que la matrice $\mathbb{S}$, car nous avons besoin de spécifier si nous parlons de la matrice $T_{1,1}$, $T_{1,2}$,... . Dans le cas de matrice $\mathbb{S}$, c'est plus simple : on a $S_{1,1} = S_{2,2}$, et $S_{1,2} = S_{2,1} = 0$.

```
void assembl_surf(Matrice& T1_1,Matrice& T1_2, Matrice& T2_1,
Matrice& T2_2, Maillage M, double* rho, double* V_S,double* V_P)
{
    int n = M.nb_face_front();
    int surf;
    for (int l =0 ; l< n ; l++)
    {
        surf = (M.Surface)[l](4);
        if(surf == 1 || surf==2)
        {
```

```
            Arete A= (M.Surface)[l];    //A est la l-ième arête de la surface maillage

            Point2D i = Point2D( M.noeud(A(1)));
            Point2D j = Point2D( M.noeud(A(2)));
            Point2D k = Point2D( M.noeud(A(3)));

            double x_i,x_j,x_k,y_i,y_j,y_k;     //on simplifie les notations
            x_i = i.Coor_x() ; x_j=j.Coor_x() ; x_k=k.Coor_x();
            y_i= i.Coor_y() ; y_j=j.Coor_y() ; y_k=k.Coor_y();

            int zone = A(5);

            double r=rho[zone-1];      //on met le rho correspondant à la zone
                                        //dans laquelle se trouve le triangle
            double VS=V_S[zone-1];
            double VP=V_P[zone-1];

          //déterminant du changement de variables
            double D= abs((x_j-x_i)*(y_k-y_i) -(x_k-x_i)*(y_j-y_i));

            int indice_ij=1;       //cas où i=j=1

            Matrice Tel= mat_surf_elem(D, r, indice_ij, surf ,VS, VP);
            //matrice élementaire


             //on met les coefficients à la bonne place dans la matrice
            T1_1(A(1),A(1)) += Tel(1,1);
            T1_1(A(1),A(2)) += Tel(1,2);
            T1_1(A(2),A(1)) += Tel(2,1);
            T1_1(A(2),A(2)) += Tel(2,2);

            indice_ij=2;    //cas où i=j=2, on refait la même chose

            Tel= mat_surf_elem(D, r, indice_ij, surf ,VS, VP);       //matrice élementaire

            T2_2(A(1),A(1)) += Tel(1,1);             //on met les coefficients à la
                                                    //bonne place dans la matrice
            T2_2(A(1),A(2)) += Tel(1,2);
            T2_2(A(2),A(1)) += Tel(2,1);
            T2_2(A(2),A(2)) += Tel(2,2);


        }
    }
}
```

Voici ici le détail des calculs de la matrice surfacique élémentaire :

```
//matrice surfacique élémentaire
```

```
//On intègre les fonctions élémentaires sur [S_i,S_j]
//indice_IJ désigne de quel TI_J on parle
//indice_surface désigne si [S_i,S_j] est un morceau du fond ou un morceau des côtés
//VP,VS désignent les valeurs propres de la matrice A

Matrice mat_surf_elem(double D, double rho, int indice_ij,
int indice_surface,double VS, double VP)       //matrice surfacique élementaire
{
    Matrice Sel = Matrice(2,2);
    Sel(1,1) =2;
    Sel(1,2)=1;
    Sel(2,1) = 1;
    Sel(2,2)=2;

    Sel*=1./6*(D*rho);

    if ( (indice_ij== 1 && indice_surface==1) || ((indice_ij== 2 && indice_surface==2)) )
        {Sel *= VP;}
    else
        {Sel *= VS;}

    return Sel;
}
```

3.3 Assemblage de F

Nous disposons des 3 matrices apparaissant dans (4) : $\mathbb{D}$ (qui est définie simplement à partir de $\mathbb{M}$, $\mathbb{K}$, et $\mathbb{T}$. Il ne nous reste qu'à assembler le vecteur F, qui représente le terme source de notre problème. On construit donc deux vecteurs F_1 et F_2, de taille N chacun, et qui contiennent, à l'indice i, la valeur de f(x_i, t), où x_i est l'abscisse du sommet i. On multiplie ensuite ces vecteurs par $\mathbb{S}$, et on obtient deux nouveaux vecteurs $F_1^{'}$ et $F_2^{'}$. On les assemble ensuite pour former un vecteur à $2N$ coordonnées.

4 Résultats

4.1 Validation

Dans un premier temps, on se contente de valider notre code sur un maillage simple, du type :

Nous prenons les valeurs de tous les paramètres égaux à 1. La fonction source est une impulsion de Dirac. Le résultat obtenu est le suivant :

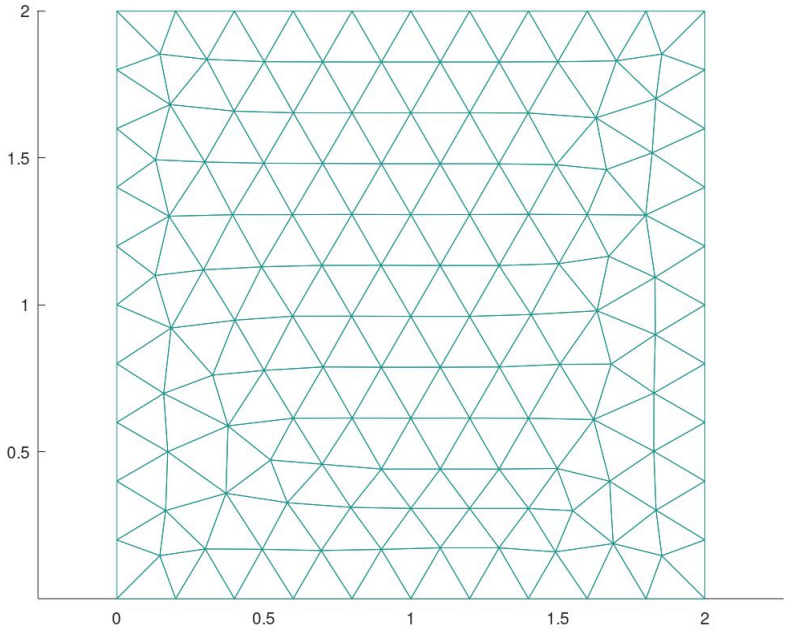


FIGURE 3 – Premier maillage sur lequel nous effectuons la validation

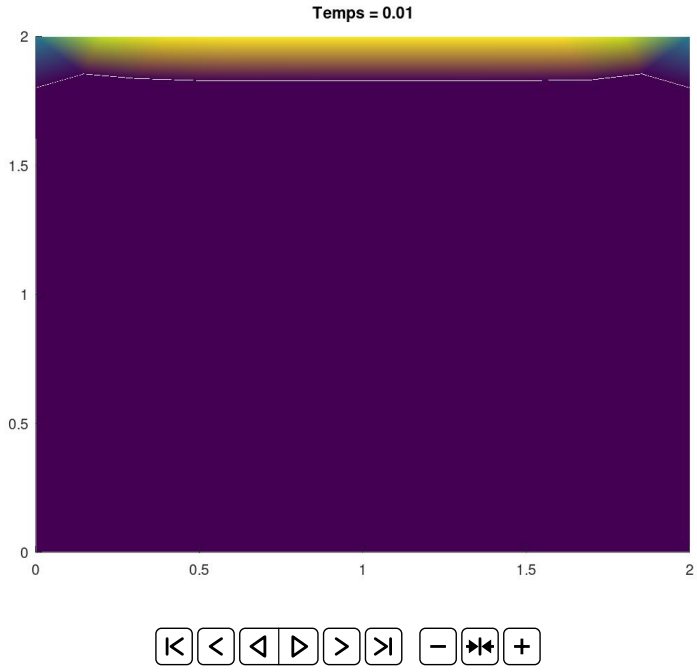


FIGURE 4 – Propagation du champ de déplacement. On a ici représenté l'intensité du champ, c'est à dire, sur chaque maille, la valeur de $\sqrt{U_k^2 + U_{k+N}^2}$.

On remarque que le code ne semble pas donner un résultat satisfaisant. Plus précisément,

1. le vecteur F semble être correct, puisque l'on voit la source au bon endroit dans le domaine.
2. on voit une légère propagation dans le domaine.

Cependant, on voit bien deux "excroissances" dans les deux coins supérieurs, donc la valeur "écrase" les autres valeurs. C'est sans doute ce qui explique que la propagation n'est que légère dans le domaine. De plus, lorsque l'on regarde bien les valeurs du fichier .txt généré par le code, on voit que les composantes de U tendent vers ∞, ce qui est impossible physiquement.

Malgré tous nos efforts, nous n'avons pas réussi à déceler l'erreur. Nous pensons qu'elle se situe dans l'assemblage de $\mathbb{K}$, mais cela se peut que ce ne soit pas le cas.

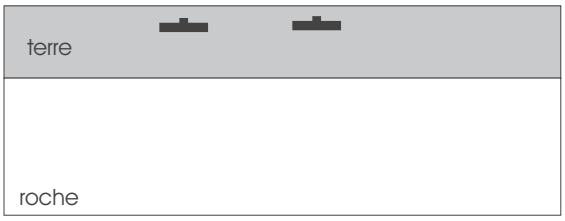
Annexes

Détection de mines
SIMNUM

2017

1 Problème

On cherche à détecter des mines enfouies au voisinage de la surface d'un milieu stratifié à l'aide d'onde élastodynamique (ultrason). Pour simplifier, on considère que le milieu est bidimensionnel et constitué d'une couche rocheuse et d'une couche de terre :



Il s'agit dans un premier temps de disposer d'un outil de simulation de la propagation d'onde dans un tel milieu. En notant respectivement Ω_t, Ω_r et Ω_m les domaines occupés par la roche, la terre et les mines, il s'agit de résoudre les équations de l'élastodynamique dans chacun des milieux $s = r, t, m$, $\vec{u}_s(x, y, t)$ désigne le champ de déplacement dans le milieu s :

$$(\mathcal{E}_s) \qquad \rho_s \frac{\partial^2 \vec{u}_s}{\partial t^2} - \mathrm{div}\sigma(\vec{u}_s) = 0 \quad \text{dans } \Omega_s$$

où le tenseur des contraintes est (λ_s et μ_s coefficients de Lamé du mileu s) :

$$\sigma_{ij}(\vec{u}) = \lambda_s \mathrm{div}\vec{u}\,\delta_{ij} + 2\mu_s \varepsilon_{ij}(\vec{u}) \quad i, j = 1, 2$$

et celui des déformations est donné par :

$$\varepsilon_{ij}(\vec{u}) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad i, j = 1, 2.$$

Aux équations précédentes, il convient d'ajouter une condition de transmission sur l'interface terre-sol qui assure la continuité du déplacement ainsi que la continuité des efforts normaux. A la surface, on supposera qu'il n'y a pas d'effort, hormis un impact localisé, ce qui se traduit par la condition limite sur Γ:

$$\sigma(\vec{u}_t).\overrightarrow{n} = \overrightarrow{f}(x, t)$$

où $\overrightarrow{f}(x, t)$ est une force donnée.
Enfin, le milieu étant non borné il est nécéssaire d'introduire des conditions aux limites sur les frontières artificielles de calcul : $\Sigma_\pm$ frontières latérales et Σ_f le "fond". Il existe plusieurs possibilités, nous retiendrons la plus simple (pas la plus efficace) qui consiste à introduire la condition absorbante du premier ordre suivante :

$$\sigma(\vec{u}).\overrightarrow{n} + \rho\mathbb{A}\frac{\partial}{\partial t}\vec{u} = 0$$
$$\text{avec } \mathbb{A} = \begin{bmatrix} V_P & 0 \\ 0 & V_S \end{bmatrix} \text{ sur } \Sigma_\pm \text{ et } \mathbb{A} = \begin{bmatrix} V_S & 0 \\ 0 & V_P \end{bmatrix} \text{ sur } \Sigma_f$$

où $V_P = \sqrt{\dfrac{\lambda+2\mu}{\rho}}$ est la vitesse des ondes de pression et $V_S = \sqrt{\dfrac{\mu}{\rho}}$ est la vitesse des ondes de cisaillement du milieu concerné.
Enfin on supposera que le milieu est au repos à l'instant initial $t = 0$.

2 Simulation directe

On inroduit le champ de déplacement total :

$$\vec{u} = \begin{cases} \vec{u}_t & \text{dans } \Omega_t \\ \vec{u}_r & \text{dans } \Omega_r \\ \vec{u}_m & \text{dans } \Omega_m \end{cases}$$

ainsi que le domaine total $\Omega = \Omega_t \cup \Omega_r \cup \Omega_m$, $\Sigma = \Sigma_+ \cup \Sigma_- \cup \Sigma_f$ et les coefficients ρ, λ et μ généraux (discontinus).
En multipliant par une fonction test $\overrightarrow{v}$ les équations volumiques et en tenant compte des conditions de transmission et des conditions aux limites, on obtient la formulation faible suivante :

$$\frac{d^2}{dt^2}\int_\Omega \rho\vec{u}.\overrightarrow{v} + \int_\Omega \lambda \mathrm{div}\vec{u}\,\mathrm{div}\vec{v}\,d\Omega + \int_\Omega 2\mu\varepsilon(\vec{u})\,\varepsilon(\vec{v}\,)\,d\Omega - \frac{d}{dt}\int_\Sigma \rho\mathbb{A}\vec{u}.\vec{v} = \int_\Gamma \overrightarrow{f}.\vec{v}$$

On cherche alors pour tout $t > 0$, $\overrightarrow{u}(t) \in H^1(\Omega)^2$ qui satisfait cette formulation pour tout $\overrightarrow{v} \in H^1(\Omega)^2$. On peut démontrer que ce problème est bien posé et qu'en outre il est dissipatif.

2.1 Schéma numérique

Afin de discrétiser cette formulation, on utilise une méthode d'éléments finis P1 en espace et un schéma saute-mouton en temps. Pour une triangulation donnée du domaine Ω constituée des sommets $(S_I)_{I=1,N}$ auxquels sont attachées les fonctions de bases $(w_I)_{I=1,N}$, on introduit les fonctions suivantes :

$$\overrightarrow{w}_I^1 = \begin{pmatrix} w_I \\ 0 \end{pmatrix} \quad \text{et} \quad \overrightarrow{w}_I^2 = \begin{pmatrix} 0 \\ w_I \end{pmatrix}$$

et l'espace $V_h = vect\left(\overrightarrow{w}_I^1, \overrightarrow{w}_I^2\right)_{I=1,N}$. Par définition des fonctions $(w_I)_{I=1,N}$ l'espace V_h est inclus dans $H^1(\Omega)^2$ et il est,de dimension $2N$. On introduit le problème semi-discrétisé : trouver $\vec{u}_h \in V_h$ tel que $\forall \overrightarrow{v} \in V_h$ on ait :

$$\frac{d^2}{dt^2}\int_\Omega \rho\vec{u}_h.\overrightarrow{v} + \int_\Omega \lambda \mathrm{div}\vec{u}_h\,\mathrm{div}\vec{v}\,d\Omega + \int_\Omega 2\mu\varepsilon(\vec{u}_h)\,\varepsilon(\vec{v}_h\,)\,d\Omega - \frac{d}{dt}\int_\Sigma \rho\mathbb{A}\vec{u}_h.\vec{v} = \int_\Gamma \overrightarrow{f}.\vec{v}$$

En introduisant le vecteur $\overrightarrow{U}(t)$ de $\mathbb{R}^{2N}$ représentant $\vec{u}_h$ sur la base $\left(\overrightarrow{w}_1^1, \overrightarrow{w}_1^2, ..., \overrightarrow{w}_N^1, \overrightarrow{w}_N^2\right)$, ainsi que les matrices $\mathbb{K}$, $\mathbb{M}$ et $\mathbb{T}$ d'ordre $2N$ définies par :

$$\begin{aligned}
(\mathbb{K}_{IJ})_{ij} &= \int_\Omega \lambda div\overrightarrow{w}_I^i\, div\overrightarrow{w}_J^j\,d\Omega + \int_\Omega 2\mu\varepsilon(\overrightarrow{w}_I^i)\,\varepsilon(\overrightarrow{w}_J^j)\,d\Omega \\
(\mathbb{T}_{IJ})_{ij} &= \int_\Sigma \rho\mathbb{A}\overrightarrow{w}_I^i.\overrightarrow{w}_J^j \\
(\mathbb{M}_{IJ})_{ij} &= \int_\Omega \rho\overrightarrow{w}_I^i\,\overrightarrow{w}_J^j\,d\Omega
\end{aligned}$$

la formulation variationnelle précédente s'écrit :

$$\mathbb{M}\frac{d^2}{dt^2}\overrightarrow{U}(t) + \mathbb{K}\overrightarrow{U}(t) - \frac{d}{dt}\mathbb{T}\overrightarrow{U}(t) = \overrightarrow{F}$$

Il s'agit d'un système différentiel sur lequel on peut appliquer le schéma de saute-mouton, $t_k = k\triangle t$ et $\overrightarrow{U}_k$ une approximation de $\overrightarrow{U}$ à l'instant t_k :

$$\mathbb{M}\frac{\overrightarrow{U}_{k+1} + \overrightarrow{U}_{k-1} - 2\overrightarrow{U}_k}{\triangle t^2} + \mathbb{K}\overrightarrow{U}_k - \mathbb{T}\frac{\overrightarrow{U}_k - \overrightarrow{U}_{k-1}}{\triangle t} = \overrightarrow{F}_k$$

avec $\overrightarrow{U}_0 = \overrightarrow{U}_1 = 0$ si on suppose que le milieu est au repos à l'instant 0.
En approchant la matrice $\mathbb{M}$ par la matrice diagonale $\mathbb{D}$ définie par condensation de masse : $\mathbb{D}_{\ell\ell} = \sum_{p=1,2N} \mathbb{M}_{\ell p}$,
le système précédent se met sous la forme explicite :

$$\overrightarrow{U}_{k+1} = 2\overrightarrow{U}_k - \overrightarrow{U}_{k-1} + \triangle t^2 \mathbb{D}^{-1} \left(\overrightarrow{F}_k - \mathbb{K}\overrightarrow{U}_k + \mathbb{T}\frac{\overrightarrow{U}_k - \overrightarrow{U}_{k-1}}{\triangle t} \right)$$

Attention ce schéma est soumis à une condition de stabilité du type CFL.

2.2 Mise en oeuvre

Il s'agit de mettre en oeuvre ce schéma numérique.
Dans un premier temps, on développera les classes et les fonctionnalités nécessaires à la méthode des éléments finis P1. On a besoin :

- de classes permettant de gérer les maillages avec en particulier une fonction de lecture de maillage généré par le logiciel *emc2* ou *gmsh* (installés sur les machines de l'ENSTA); travailler avec le format amdba ou msh.
- de classes vecteurs permettant de réaliser tous types de combinaison linéaire
- de classes matrices stockées profils (voir annexe du poly MA201) permettant de stocker les matrices éléments finis, ainsi que du produit matrice-vecteur. Attention, à chaque sommet S_I du maillage, est attaché deux degrés de liberté correspondant aux deux composantes (u_1^I, u_2^I) du champ de déplacement. On a le choix entre deux types d'ordonnancement des degrés de liberté : soit en regroupant toutes les composantes du même type ensemble $(u_1^1, u_1^2, ...u_1^N, u_2^1, u_2^, ...u_2^N)$ soit en regroupant les composantes 2 par 2 $(u_1^1, u_2^1, ...u_1^N, u_2^N)$. Dans ce dernier cas, le profil de la matrice doit être adapté. Dans le premier cas, la matrice peut être considérée comme une matrice par blocs (2x2), le profil de chaque bloc étant le profil usuel d'une matrice élément fini. Je vous conseille d'utiliser cette approche.

On se référera au cours d'éléments finis MA201 pour implémenter le calcul des matrices éléments finis.
On écrira des fonctions de sortie sur fichier du champ de déplacement aux différents instants permettant de représenter sous Matlab le champ de déplacement, le champ de contrainte principales, sous forme statique ou dynamique (film).

2.3 Validation

On validera le programme en diverses étapes :

- test des fonctionnalités principales (lecture du maillage, combinaison de vecteurs, calcul des matrices éléments finis, produit matricexvecteur, sorties sur fichier)
- test sur un cas très simple : un seul milieu (pas de stratification, ni mine)
- test avec un milieu et une mine
- test avec un milieu straifié et une mine ou plusieurs mines

Attention à bien choisir les données des matériaux, de la donnée f. A cet effet, on pourra considérer des données de type impulsion ou des données harmoniques.

3 Organisation du travail (3p)

On pourra se partager le travail de la façon suivante :

- Un élève prend en charge le développement des classes utilitaires (vecteur, matrice **creuse**, maillage, point, triangle). La définition de ces classes et les fonctionnalités de base doivent être fixée rapidement pour que les autres puisse savoir sur quoi ils peuvent s'appuyer. Le développement des fonctionnalités de base (création, destruction, accès) doit être réalisé rapidement. Dans un deuxième temps, on développera les fonctionnalités avancées (produit matrice-vecteur en particulier)
- Un deuxième élève prendra en charge la fabrication des matrices éléments finis (calcul élémentaire, assemblage)
- Le dernier élève pourra s'occuper de développer les outils de lecture de maillage (prioritaire) ainsi que ceux de visualisation (sortie sur fichier et exploitation sous Matlab) et s'occupera du programme principal.

4 Extension

Si la simulation directe fonctionne correctement, on pourra envisager de développer une méthode simple de détection d'une mine basée sur l'écho de l'onde induit par la mine. Il s'agit de déterminer dans un premier temps la donnée la plus pertinente puis de "mesurer", par exemple au niveau de la zone d'émission, les signaux qui reviennent et d'en déduire la distance de la mine connaissant les vitesses de propagation des ondes.