Représentation Temps-Fréquence : travaux pratiques

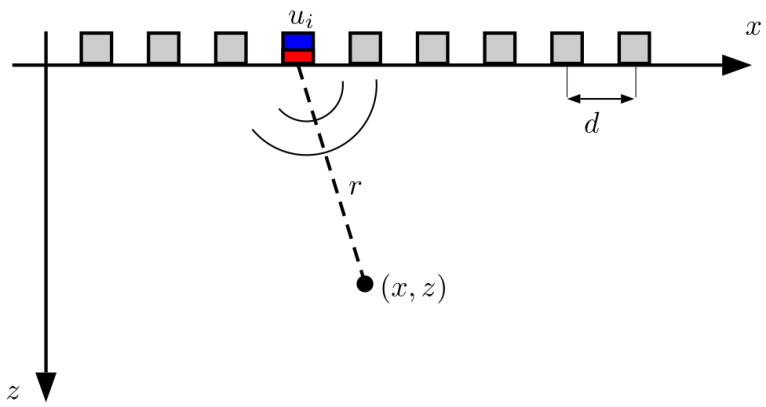
Yassine Jamoud, Samy Haffoudhi

11 février 2022

1 Présentation du TP

Ce TP concerne l'imagerie Synthetic Apperture Focusing Technique (SAFT) pour le contrôle par ultrasons. Nous nous intéressons au contrôle d'un bloc d'aluminium contenant des petits trous à l'aide d'une sonde multi-élément en contact direct avec la pièce. Nous mettrons en œuvre une méthode d'imagerie dite temporelle que nous commencerons par implémenter dans matlab à l'aide de boucles `for` avant de convertir ce code en fonction MEX afin de comparer le temps de calcul des deux mises en œuvre.

FIGURE 1 – Principe de l'imagerie SAFT

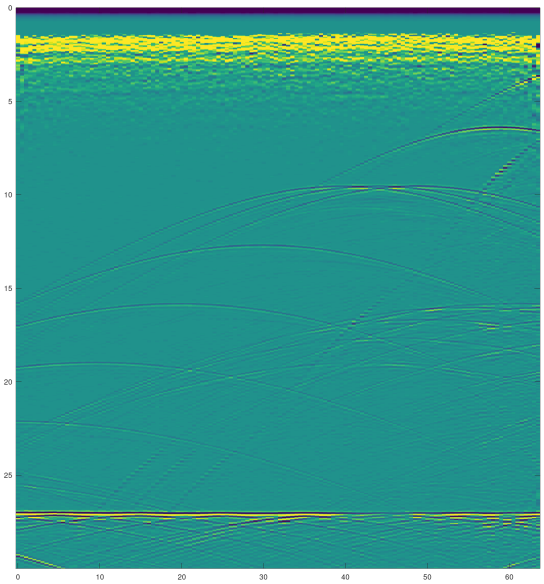


2 Ouverture du fichier

— On a $N_t = 1500$ et $N_{el} = 128$.

— Affichons l'image des données en fonction de t et u :

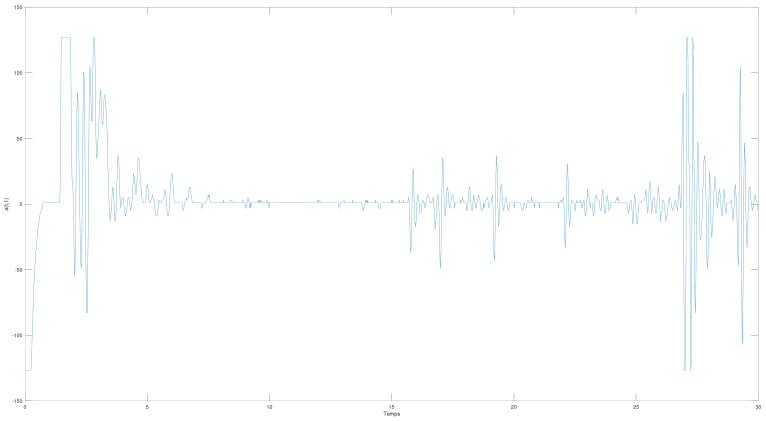
FIGURE 2 – Image des données



— Cette image n'est pas exploitable, on est incapable d'y distinguer les petits trous du bloc d'aluminium.

— Affichons maintenant un Ascan :

FIGURE 3 – Ascan



— On déduit de la figure précédente que $log_2(2 \times 128) = 8$ bits sont utilisés.

3 Définition de la grille de reconstruction

Nous modifions le fichier `main.m` disponible en annexe.

4 Reconstruction de l'image par une méthode temporelle

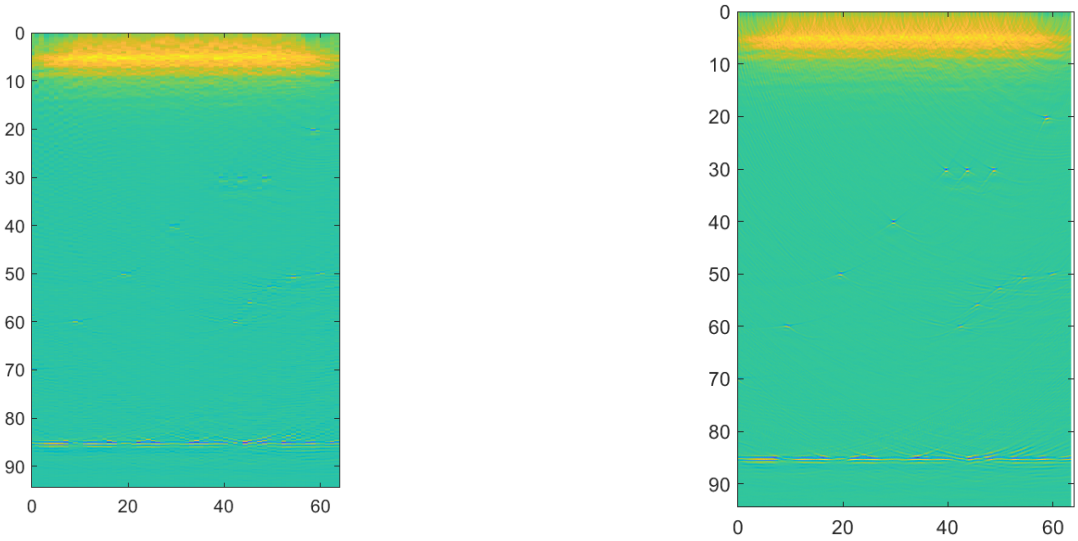
— D'après la figure 2, on a : $r(x, z, u_i) = \sqrt{(u_i - x)^2 + z^2}$.

— On a : $\tau(x, z, u_i) = \frac{2r(x,z,u_i)}{c}$.

— Donc, on peut calculer $O(x, z) = \sum_{i=1}^{N_{el}} a(\tau(x, z, u_i), i)$.

— On ajoute la procédure au fichier `main.m` pour le calcul de O.

— On considère $N_x \in \{64, 128, 256, 512, 1024, 2048\}$.
Affichons deux images obtenues :



(a) $N_x = 64$ (b) $N_x = 2048$

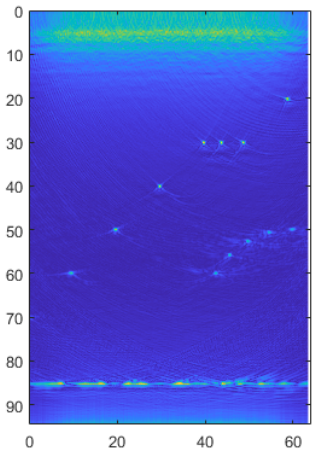
FIGURE 4

On observe que les trous sont bien plus faciles à distinguer sur la figure correspondant à $N_x = 2048$ que sur la première, correspondant à $N_x = 64$.
Les temps de calcul associés sont respectivement : 0.27s, 0.43s, 0.86s, 1.75s, 3.38s et 6.74s.
Ainsi, on observe que la complexité est linéaire en N_x.

5 Post-traitement de l'image

Affichons une image O obtenue après post-traitement :

FIGURE 5 – Image O après post-traitement



On observe alors que le post-traitement a permis de gagner en lisibilité en jouant sur les couleurs. En effet, les trous de la plaque ressortent beaucoup plus du fond.

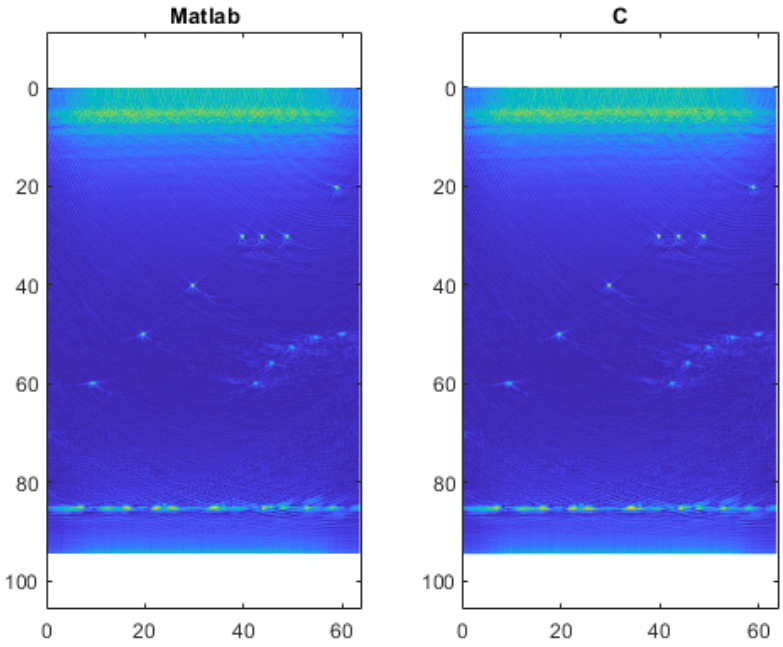
6 Accéleration avec une fonction MEX

Le contenu du fichier `MEX_SAFT.c` est disponible en annexe.

7 Comparaisaon de deux mises en oeuvre

— Commençons par afficher une des images obtenues pour chaque version :

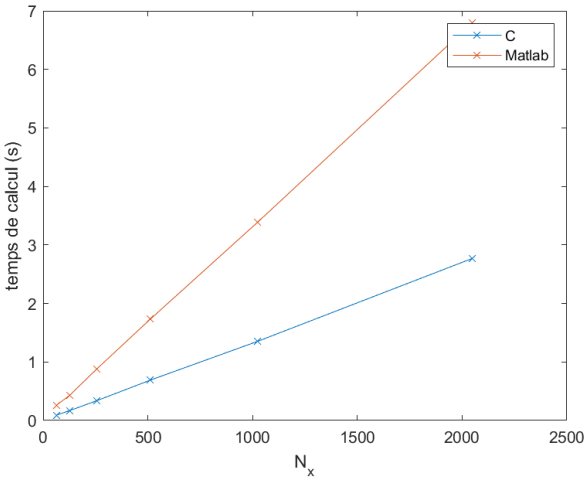
FIGURE 6 – Deux images O



Ces images semblent être identiques, c'est le cas pour toutes les valeurs de N_x.

— Affichons le temps de calcul des deux versions :

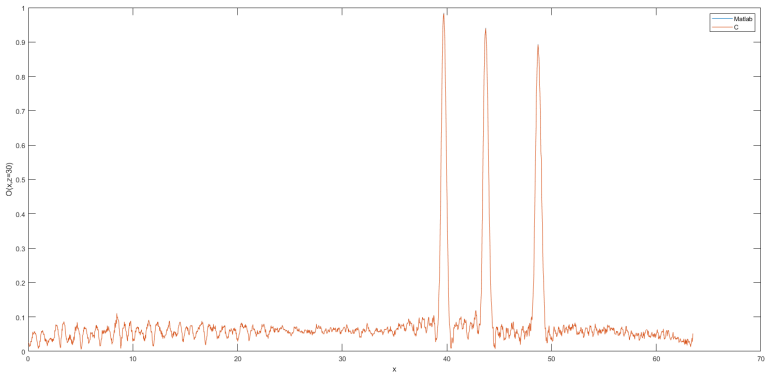
FIGURE 7 – Temps de calcul



La complexité est évidemment inchangée entre les les deux versions mais la version en MEX est environ 60% plus rapide que la version Matlab. Cette différence est entièrement due aux différences entre les deux langages. Le langage C, étant bas niveau, est connu pour sa rapidité.

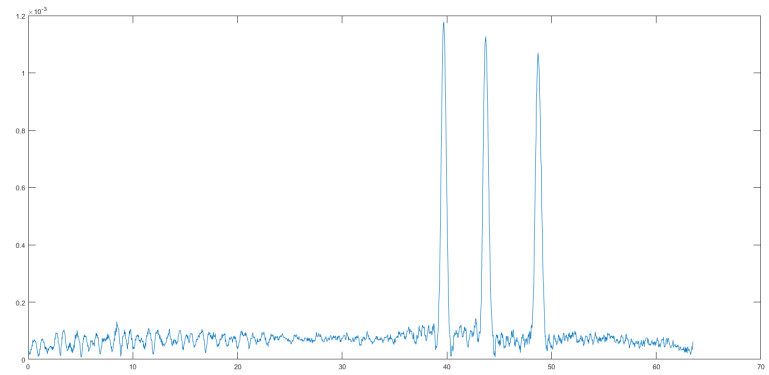
— Affichons la ligne $z = 30$

FIGURE 8 – Ligne $z = 30$



Les trous dans la direction x sont espacés d'environ 5 cm.
Affichons enfin la différence :

FIGURE 9 – Différence entre les deux signaux



Cette différence n'est pas nulle. Il y alors une différence entre les deux signaux. En effet, en les observant de plus près, on note la présence d'un léger retard entre ces signaux.

Conclusion

Ainsi, lors de ce TP nous avons pu implémenter une méthode d'imagerie temporelle afin d'observer les petits trous de la plaque. Nous avons pu comparer les performances de la procédure entre une version matlab et MEX. On a alors conclu que la version MEX était environ 60% plus rapide que son équivalent Matlab.

A Main.m

```
clear all
close all

load('dataSAFT_exp.mat')

[N_t, N_el] = size(A);

t = linspace(0,(N_t-1)/Fs, N_t);
u = linspace(0, (N_el-1)*d, N_el);

% figure()
% imagesc(u,t,A)

% figure()
% plot(t, A(:,1))
% xlabel('Temps')
% ylabel('a(t,1)')

N_x = [64, 128, 256, 512, 1024, 2048];
x_min = u(1); x_max = u(end);
dz = c/(2*Fs);
z_min = t(1); z_max = (N_t-1)*dz;
z = z_min:dz:z_max;
N_z = length(z);

tps_matlab = zeros(length(N_x),1);
tps_c = zeros(length(N_x),1);

for l = 1:length(N_x)
    x = linspace(x_min,x_max,N_x(l));
    O = zeros(N_z,N_x(l));

    tic
    for i = 1:N_x(l)
        for j = 1:N_z
            s = 0;
            for k = 1:N_el
                tau = (2/c)*sqrt((x(i)-u(k))^2+z(j)^2);
                if round(Fs*tau)+1<N_t
                    s = s + A(round(Fs*tau)+1, k);
                end
            end
            O(j, i) = s;
        end
    end
    tps_matlab(l) = toc;

    O = abs(hilbert(O));
    O = O/max(O(:));

    tic
    O_c = MEX_SAFT(c, Fs, N_t, N_el, u, N_x(l), N_z, x, z, A);
    tps_c(l) = toc;

    O_c = reshape(O,[N_z,N_x(l)]);
    O_c = abs(hilbert(O));
    O_c = O/max(O_c(:));

```

```
    figure()
    subplot(1,2,1)
    imagesc(x,z,O); axis equal; xlim([0 round(x(end))])
    title('Matlab')

    subplot(1,2,2)
    imagesc(x,z,O_c); axis equal; xlim([0 round(x(end))])
    title('C')
end

figure()
plot(N_x,tps_c, 'x-');
hold on
plot(N_x,tps_matlab, 'x-');
legend('C', 'Matlab')
xlabel('N_x')
ylabel('temps de calcul (s)')

z_ind = 478;
figure()
plot(x,O(z_ind,:))
hold on
plot(x,O_c(z_ind,:))
legend('Matlab', 'C')
xlabel('x')
ylabel('O(x,z=30)')

figure()
plot(x, abs(O(z_ind,:) - O_c(z_ind,:)))
xlabel('x')
ylabel('Diff rence')
```

B MEX_SAFT.c

```
#include "mex.h"
#include "math.h"

/* La fonction de calcul */
void MEX_SAFT(double c, int Fs, int Nt, int Nel, double *p_u, int Nx, int Nz, double
    *p_x, double *p_z, double *p_A, double *p_O)
{
    double tau;
  int s,ind_tau;

    for (int i = 0; i < Nx; i++) {
        for (int j = 0; j < Nz; j++) {
            s = 0;
            for (int k = 0; k < Nel; k++) {
                tau = (2 / c) * sqrt((p_x[i] - p_u[k]) * (p_x[i] - p_u[k]) + p_z[j]
    * p_z[j]);
                ind_tau = (int) floor(Fs*tau+0.5);
                if (ind_tau < Nt) {
                    s += p_A[ind_tau + k * Nt];
                }
            }
            p_O[j + i*Nz] = s;
        }
    }
}
```

```
24
25 /* La fonction d'appel */
26 void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
27 {
28     /*  D clarations  variables C */
29     double c;
30   int Fs;
31   int Nt;
32   int Nel;
33   double *p_u;
34   int Nx;
35   int Nz;
36   double *p_x;
37   double *p_z;
38   double *p_A;
39   int NoutSize;
40   double *p_O;
41
42     /* Affectation des entres */
43     c = mxGetScalar(prhs[0]);
44     Fs = mxGetScalar(prhs[1]);
45     Nt = mxGetScalar(prhs[2]);
46     Nel = mxGetScalar(prhs[3]);
47     p_u = mxGetPr(prhs[4]);
48     Nx = mxGetScalar(prhs[5]);
49     Nz = mxGetScalar(prhs[6]);
50     p_x = mxGetPr(prhs[7]);
51     p_z = mxGetPr(prhs[8]);
52     p_A = mxGetPr(prhs[9]);
53
54     /* Affectation des sorties */
55     NoutSize = Nx*Nz;
56     plhs[0] = mxCreateNumericMatrix((mwSize)NoutSize,1,mxDOUBLE_CLASS,0);
57     p_O = mxGetData(plhs[0]);
58
59     /* Appel de la fonction de calcul */
60     MEX_SAFT(c,Fs,Nt,Nel,p_u,Nx,Nz,p_x,p_z,p_A,p_O);
61 }
```