Location Algorithm for Source of Acoustic Emission in Non-convex Concrete Elements

Revisition of source code
and abridged version of the original report

In english

Mika Lindström
Dec MMXX

The following concerns about my stydy in University of Eastern Finland (UEF) about Acoustic Emission in concrete structures. The original report, written in Finnish, is available at UEF and is gainable with a special request. If you wish to request it, please inform either kirjaamo@uef.fi or mika.lindstroem@gmail.com. In your request, please give your full name, phone number, educational level, alma mater, employer and the need for the report, which may be as little as interest towards it.

As a revisition and abridged version, please refer to all latter sources every time something is claimed in this text. I will omit source numbering, but they are available in the original report.

The source code is fully revisited by me, and should now be reader friendly. At the time of original report, COMSOL's usage of parameters as literal <u>strings</u>, COMSOL's object oriented (Java) way and MATLAB's preferenced parameter usage of IEEE-754 <u>double</u> caused confusion on my behalf. But these problems should have went over with better understanding and hours after hours of programming. MATLAB's functions **num2str()** and **strcat()** were helpful. MATLAB is also capable of object oriented programming, see:
https://se.mathworks.com/company/newsletters/articles/introduction-to-object-oriented-programming-in-matlab.html for further information.

The revisited code is written with MATLAB R2014b and COMSOL 5.1. It may, or may not work with different versions. Please follow complier instructions and make changes to the code if you wish use different versions of these two programs. It is also possible to use Java instead of MATLAB, but even there, version incomplabilities can cause harm to the user. As of COMSOL version 5.3, there exist a developer tab to help the purpose of programmer. See: https://www.comsol.com/release/5.3/comsol-multiphysics to more information. Third option is to save model as Visual Basic for Applications file and run in that enviroment. The saved code looks at least as straightfoward as with the other two options.

Contents

1 Introduction

Concrete is an important material for structures. And it is also important to understand it's health status as structures gain age.

Fractures in concrete produce ultrasounds, about in range of 20 kHz – 1 MHz. These ultrasounds are measurable on the surface of the concrete structure with special piezo-electric sensors. These sensors can be connected to a computer system and the situation can be analyzed, see A. Tobias from refences for his ideas.

Location of the fracture is based on time-of-flight equation, which for N amount of sensors is

$$t^i = t^0 + D(x^i, x^0)/c \ \ ||i \in 1...N, \tag{1}$$

where t^i is time when soundwave reaches sensor with label i, t^0 is time when fraction has happened, D is minimum-energy-metric of the concrete element under analysis, x^i is sensor's, labelled with i, position, x^0 is the position of fracture and c speed of (ultra)sound in concrete.

For convex elements, the minimum-energy-metric is the same as euclidian metric d

$$d(x, y) = \sqrt{\sum_{i=1}^{3} (x_i - y_i)^2}. \tag{2}$$

For non-convex elements the minimum-energy-metric is not easily calculable, or it may even be that it's not even describeable in closed form. This text's solution is to use a search method to solve this inversion problem. The solution is based on forward-modelling repeatly with COMSOL 5.1 simulations, on Gauss-Newton algorithm and on perturbation method, where differential are approximated numerically with small positive number epsilon.

2 Locating Acoustic Emission Source, Fracture

The algorithm transforms data, i.e times when a sensor registers larger than threshold value pressure wave, to differences of these times. Then one of to be estimated parameters, namely t^0 i.e. time of fracture incident drops out. This parameter can be considered unnecessary in building structures, where the acoustic emission singal is in few milliseconds in sensor, but not in seismology as earthquake's true event time is a special intrest. As N amount of sensors have pair amount Comb(N,2) i.e.

$$\binom{N}{2} = \frac{N!}{2!(N-2)!} = \frac{N(N-1)}{2}, \tag{3}$$

so for example 4 sensors have 6 pairs. Of these pairs we can make a combination matrix A, which, for example, could be in this form:

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 2 & 3 \\ 2 & 4 \\ 3 & 4 \end{pmatrix}. \tag{4}$$

For the fracture location solving Gauss-Newton algorithm, the data is set to time-differences

$$z_i = t^{A_{i1}} - t^{A_{i2}} \quad || \quad i \in \{1, ..., N(N-1)/2\}, \tag{5}$$

where $t^{A_{ij}}$ time of wave registered at sensor, labelled with A_{ij}. The modelling function to the Gauss-Newton algorithm is

$$h(x^0)_i = (D(x^0, x^{A_{i1}}) - D(x^0, x^{A_{i2}}))/c, \quad || \quad i \in \{1, ..., N(N-1)/2\}, \tag{6}$$

where x^0 is the position of fracture and $x^{A_{ij}}$ sensor's, labelled with A_{ij} location. The values for the model function $h(x^0)$ are gained with COMSOL Multiphysics version 5.1's pressure acoustics simulation. The simulations wave equation is

$$\frac{1}{\rho c^2}\frac{\partial^2 p}{\partial t^2} - \nabla \cdot \left(\frac{\nabla p}{\rho}\right) = Q_m, \tag{7}$$

where ρ on density on concrete, c speed of sound in concrete, p is location and time spesific pressure and Q_m a source-term. The signal is registered when certain threshold is gained, for example 50 MPa. The source-term Q_m is modelled as a point source with equation

$$Q_m = \frac{\partial Q_s}{\partial t}\delta(x - x^0), \tag{8}$$

where

$$Q_s = ae^{-\pi^2 f_0^2 (t-t_p)^2} \tag{9}$$

is Gaussian bell-curve ja f_0 frequency of the wave. The boundary conditions have not been changed from COMSOL's default settings in revisited version.

We need to find fracture location x^0. To attend this we try to minimize functional

$$||z - h(x^0)||^2. \tag{10}$$

The Gauss Newton algorithm suits this purpose. The algorithm needs a Jacobian matrix and it is in form of

$$J(x^0) = \left(\frac{\partial h(x^0)}{\partial x_1^0} \quad \frac{\partial h(x^0)}{\partial x_2^0} ... \right). \tag{11}$$

For convex element, the entries of this matrix is calculable and the are:

$$J(x^0)_{ij} = \left(\frac{x_j^0 - x_j^{A_{i1}}}{c \cdot d(x^0, x^{A_{i1}})} - \frac{x_j^0 - x_j^{A_{i2}}}{c \cdot d(x^0, x^{A_{i2}})} \right) || \quad i \in \{1, ..., N(N-1)/2\}, \quad j \in \{1, 2, ...\} \tag{12}$$

For non-convex concrete structure element, we calculate these entries numerically:

$$\frac{\partial h(x^0)}{\partial x_j^0} = (h(x^0 + \epsilon e_j) - h(x^0))/\epsilon \quad || \quad j \in \{1, 2, ...\}, \tag{13}$$

missä ϵ is a small positive number epsilon ja e_j unit vector toward direction j. The iterative Gauss-Newton algorithm, which we hope to locate x^0 is

$$x^{0,k+1} = x^{0,k} + \alpha^k J(x^{0,k})^\dagger (z - h(x^{0,k})) \tag{14}$$

where $J(x^{0,k})^\dagger$ is Jacob's matrix' $J(x^{0,k})$ Moore-Penrose's pseudoinverse ja α^k is a step parameter.

3 Program

The program files should be found with the same repository where this text was. The programs are also given in appendix later in this text.

The task started with opening COMSOL 5.1 desktop application. New model is selected with model builder. Time depented 2D pressure acoustic model is choosed. Material (concrete) is selected from COMSOL's material library. If either density or speed of sound are not defined by material library, please update it with literature value. A square with one squaremeter area is drawn. A circle is drawn inside this square. This circle is differenciated from the square. Dimensions does not matter in this point, they will be later changed by MATLAB code. A point is put somewhere inside the concrete, but not in the circular void. This point is the fracture source point. Gaussian source is selected to that point. Few probes, for example four, are put to boundaries of the element. Model is runned once. The element should change colour, these are changes in pressure and the sound should echo there some time. Now let us compact history. Choose **File** → **Compact History**. This is a crucial step as COMSOL keeps track of everything but we need a clean file. Now run file even once and **export** probe table, for example as **probetable.csv**. Save the model **Save As**, choose MATLAB file and name it, for example, as **comsol_model.m**.

3.1 Preparing comsol_model.m

This **.m** file must be prepared to accept parameters. The first row can be changed to

```
function comsol_model(fault_location,parameters,sensor_locations)
```

and last line

```
out=model
```

removed. The file should have lines like these:

```
model.geom('geom1').create('pt1', 'Point');
model.geom('geom1').feature('pt1').setIndex('p', '-0.35', 0, 0);
model.geom('geom1').feature('pt1').setIndex('p', '-0.2', 1, 0);
model.geom('geom1').run;
model.geom('geom1').run('fin');
```

This list of lines creates our sourcepoint and sets it into location (-0.35,-0.2) and runs the geometry again. Let us change this to accept our own parameters instead of constant point.

```
model.geom('geom1').create('pt1', 'Point');

point_x_string=num2str(fault_location(1));
point_y_string=num2str(fault_location(2));

model.geom('geom1').feature('pt1').setIndex('p', point_x_string, 0, 0);
model.geom('geom1').feature('pt1').setIndex('p', point_y_string, 1, 0);
model.geom('geom1').run;
model.geom('geom1').run('fin');
```

For multiple similar objects, a loop is used:

```
size_of_sensor_locations=size(sensor_locations);

for i=1:size_of_sensor_locations(2)
    domstring=strcat('pdom',num2str(i));
    pstring=strcat('ppb',num2str(i));
    x_string=num2str(x(1,i));
    y_string=num2str(x(2,i));
    model.probe(domstring).setIndex('coords2', x_string, 0, 0);
    model.probe(domstring).setIndex('coords2', y_string, 0, 1);
    model.probe(domstring).feature(pstring).set('window', 'window1');

end
```

That replaced COMSOL auto-generated lines:

```
model.probe('pdom1').setIndex('coords2', '0', 0, 0);
model.probe('pdom1').setIndex('coords2', '0.5', 0, 1);
model.probe('pdom1').feature('ppb1').set('window', 'window1');
model.probe('pdom2').setIndex(...
...
model probe('pdom7').feature('ppb7').set('window', 'window1');
```

Data table is important to export, so ensure your code has similar following lines:

```
result_path=strcat(pwd,'\probedata.csv');
model.result.table('tbl1').save(result_path);
```

When testing your program, you can save model as **.mph** file which can be opened with COMSOL application for visual inspection:

```
model_path=strcat(pwd,'\physicalmodel.mph');
model.save(model_path)
```

Please, comment out previous lines when you do not need visual model, as saving model can take unnecessary time when running main algorithm. To better try prevent inverse crime, i.e. situation where simulation and solution is made with same system, one could try and change values of COMSOL FEM-mesh and fracture frequency to different values at data collection and solution algorithm, as was case with original code:

```
l = c/f; % lambda (wavelength)
hmax = l/lph;
hmax_string=num2str(hmax);


model.mesh('mesh1').feature('size').set('custom', 'on');


model.mesh('mesh1').feature('size').set('hmax', hmax_string);
```

3.2 Writing main.m

The **main.m** is, as it's name says, main script file. It builds necessary mathematics of the algorithm and calls **comsol_model.m** script as it's servant. The program's hierarchy is shown in figure 1. When **main.m** calls **comsol_model.m**, a data file, **probedata.csv**, is made, or if already exists, updated. Then **main.m** reads this data and uses it to update it's algorithm. First my main script defines parameters and sensor locations. Then it starts it's loop cycle and builds up a visualisation in a figure window. There are two function files too, **getz.m**, which gives a direct command to **comsol_model.m**, reads **probedata.csv** and with help of **difz.m**, a combination matrix builder, outputs a data vector of time-differences, directly usable for Gauss-Newton algorithm. Figure 2 shows one output with eight sensors. All script files are after reference list as an appendix.

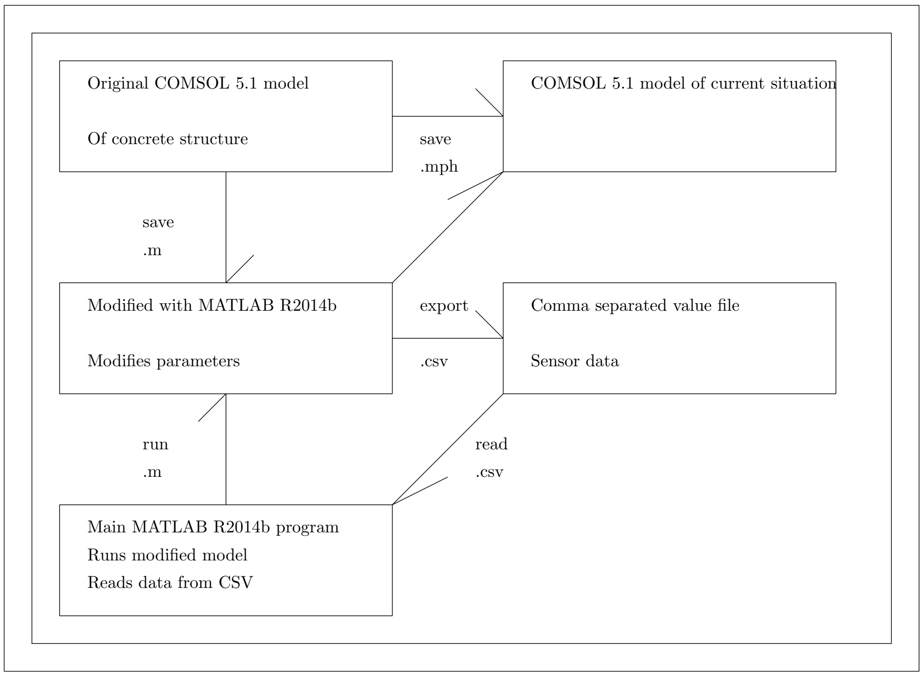
Original COMSOL 5.1 model
Of concrete structure
save
.mph
COMSOL 5.1 model of current situation
save
.m
Modified with MATLAB R2014b
Modifies parameters
export
.csv
Comma separated value file
Sensor data
run
.m
read
.csv
Main MATLAB R2014b program
Runs modified model
Reads data from CSV

Figure 1: Program's hierarchy.

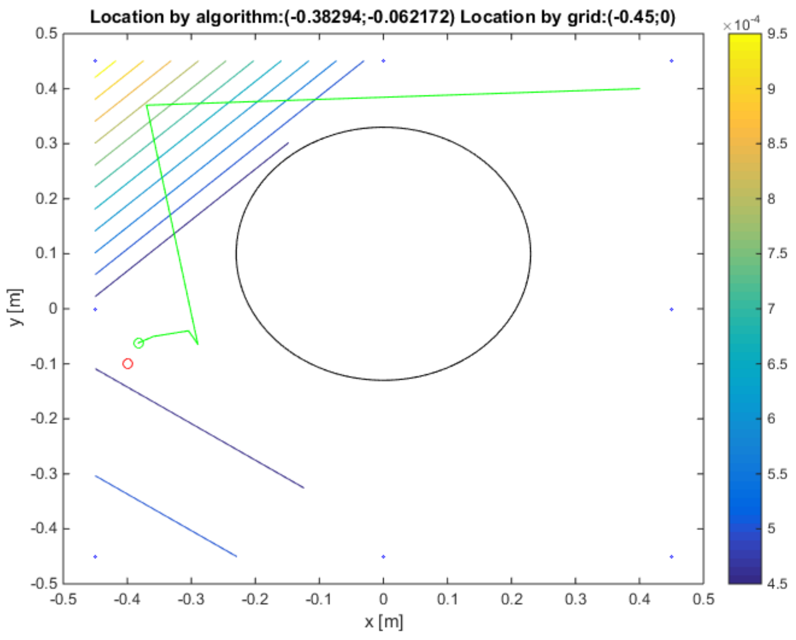

Figure 2: Output of main.m with grid constant 3.

References

[1] Landis E, "Micro-macro fracture relationships and acoustic emissions in concrete". Construction and Building Materials 13 (1999) 65-72

[2] Ohtsu M, Shigeishi M, "Nondestructive evaluation of defects in concrete by quantitative acoustic emission and ultrasonics". Sakata Ultrasonics 36 (1998) 187- 195

[3] Baxter M, ym., "Delta T source location for acoustic emission". Mechanical systems and signal processing 21.3 (2007) 1512-1520.

[4] Grosse C, Ohtsu M, "Acoustic Emission Testing". Springer-Verlag (2008).

[5] Tobias A, "Acoustic-emission source location in two dimensions by an array of three sensors". Non-destructive testing 9.1 (1976) 9-12.

[6] Colton D, Rainer K, "Inverse acoustic and electromagnetic scattering theory". Vol. 93. Springer Science & Business Media, (2012).

4 Program files

4.1 main.m

```
%calculates failure location in concrete structure with internal circ void
%uses perturbation+gauss-newton numerical algorithms

format long;

%sensor locations (any amount,but more the better and too much is too much)



 x=[  0     0.5        0    -0.5     0.5    -0.5    -0.5     0.5;%x
    0.5       0    -0.5       0     0.5    -0.5     0.5    -0.5];%y


%location of the hole and radius
hole_radius=0.23;%0.15;
hole_loc_x=0;
hole_loc_y=0.1;

hole_loc_vec=[hole_loc_x; hole_loc_y;];

%grid constant for calculating contours
gridc=3;



%iterations
k=5;



lph=10;%element per wavelenght
%freq in data
```

```
f=25000;
%pulse speed
c=3200;
%model 3=circle off system
pressure_threshold=0.5e8;%pressure where signal is registered
parameters=[f,c,pressure_threshold,lph,hole_radius,hole_loc_x,hole_loc_y];



lambda=c/f;%wavelength


%true damage location(random maybe)

%loc=0.8*[rand rand]'-0.4;%range=-0.4 to 0.4, 0.5 is too near to surface
loc=[-0.4 -0.1]';





%get z i.e data from sensors
z=getz(loc,parameters,x);


%lph->6 and f->10000 to prevent inverse crime, update parameters
lph=6;
%freq in inversion
f=25000;
parameters=[f,c,pressure_threshold,lph,hole_radius,hole_loc_x,hole_loc_y];
%step lenght
alpha=1;
%perturbation epsilon=a*lambda+b
epsa=0;%1/6;%0.1
epsb=0.065;%0.125
eps=epsa*lambda+epsb;
%breaking constant
s=0.5;

history_of_iterations=zeros(2,k);



NN=size(x);%amoutn of sensors
NN=NN(2);
n=[0.4 0.4]';%initial guess to algorithm



J1=zeros(NN,1);
J2=zeros(NN,1);

fig1=figure;
%fig2=figure;


figure(fig1);



t = linspace(0,2*pi);
plot(hole_loc_x+hole_radius*cos(t),hole_loc_y+hole_radius*sin(t),'color','black')
hold on
```

```
scatter(loc(1),loc(2),'red')
axis([-0.5 0.5 -0.5 0.5 ])
hold on
drawnow
```

```
    h=zeros(NN,1);

A=linspace(-0.45,0.45,gridc);
 B=linspace(-0.45,0.45,gridc);
  [A,B]=meshgrid(A,B);
    C=zeros(gridc,gridc);


%%%build meshgrid C
for ix=1:gridc
    for iy=1:gridc
        nn1=[A(ix,iy) B(ix,iy)]';

        if (norm(nn1-hole_loc_vec)<hole_radius )
            C(ix,iy)=NaN;%NaN if inside void
        else

            figure(fig1);


            scatter(nn1(1),nn1(2),2,'blue')
            drawnow
            hold on


            h=getz(nn1,parameters,x);
            fun=norm(z-h);
            C(ix,iy)=fun;

        end

    end
end

figure(fig1);

contour(A,B,C)
colorbar

hold on

scatter(loc(1),loc(2),'red')
drawnow
```

```
for index=1:k


%update h

h=getz(n,parameters,x);
%check if forward or backward difference due limitations

varx=1;
vary=1;

%perturbation
nx=n+varx*[eps; 0];
ny=n+vary*[0; eps];

hx=getz(nx,parameters,x);
hy=getz(ny,parameters,x);




%update J (jacobian)
J=[varx*(hx-h)/eps    varx*(hy-h)/eps ];




%G-N iteration step
lastn=n;%remember last position

% n=n+alpha*pinv(J)*(z-h);
step=pinv(J)*(z-h);


%this cond does not accept too small step at beginning
if norm(step)<2*(eps/k)
    step=eps*[rand rand]';
end

%update location
n=n+step;







%geometrical limitations goes here

%angle relative to center of void
phi=angle((-hole_loc_x+n(1))+((-hole_loc_y+n(2))*1i));
```

```
%if going too near void, push away
if norm(n-hole_loc_vec)<hole_radius+eps/2
    n(1)=hole_loc_vec(1)+(hole_radius+eps*2)*cos(phi+2*pi*20/360);
    n(2)=hole_loc_vec(2)+(hole_radius+eps*2)*sin(phi+2*pi*20/360);
end

%if too near edges
if n(1)>0.5-eps/2
    n(1)=0.5-eps*2;
end


if n(2)>0.5-eps/2
    n(2)=0.5-eps*2;
end

if n(1)<-0.5+eps/2
    n(1)=-0.5+eps*2;
end

if n(2)<-0.5+eps/2
    n(2)=-0.5+eps*2;
end









history_of_iterations(:,index)=n;
figure(fig1);
line([n(1) lastn(1)], [n(2) lastn(2)],'Color','green')
hold on
drawnow


end



scatter(n(1),n(2),[],'green')

axis([-0.5 0.5 -0.5 0.5 ])
hold on



xlabel('x [m]');
ylabel('y [m]');

titlestr=strcat('Location by algorithm:(',num2str(n(1)),';', num2str(n(2)),')');


titlestr=strcat(titlestr,' Location by grid:(',num2str(A(C==min(min(C)))),';',num2str(B(C==min(mi
title(titlestr);
```

4.2 comsol_model.m

```
function comsol_model_improved1(loc,parameters,x)

%outputs sensor data as csv files
%loc=fault location, x=sensor locations
size_of_x=size(x);
f=num2str(parameters(1));%frequncy
c=num2str(parameters(2));%speed of sound in concrete

lph = parameters(4); % elements per wavelength

%internal void parameters (2string as comsol wants):
hole_radius=num2str(parameters(5));
hole_loc_x=num2str(parameters(6));
hole_loc_y=num2str(parameters(7));

%fault location
fault_x=num2str(loc(1));
fault_y=num2str(loc(2));

matlab_path_string=pwd;




%
% comsol_model_aftertest.m
%
% Model exported on Dec 25 2020, 20:22 by COMSOL 5.1.0.234.

import com.comsol.model.*
import com.comsol.model.util.*

model = ModelUtil.create('Model');

model.modelPath(matlab_path_string);

model.label('testrun2.mph');

model.comments('Concrete block with void');

model.modelNode.create('comp1');

model.file.clear;

model.geom.create('geom1', 2);

model.mesh.create('mesh1', 'geom1');

model.geom('geom1').create('sq1', 'Square');
model.geom('geom1').feature('sq1').set('pos', {'-0.5' '-0.5'});


model.geom('geom1').create('c1', 'Circle');
model.geom('geom1').feature('c1').set('r', hole_radius);
model.geom('geom1').feature('c1').set('pos', {hole_loc_x hole_loc_y});
model.geom('geom1').create('dif1', 'Difference');
model.geom('geom1').feature('dif1').selection('input').set({'sq1'});
model.geom('geom1').feature('dif1').selection('input2').set({'c1'});
model.geom('geom1').create('pt1', 'Point');
model.geom('geom1').feature('pt1').setIndex('p', fault_x, 0, 0);
model.geom('geom1').feature('pt1').setIndex('p', fault_y, 1, 0);
model.geom('geom1').run;
model.geom('geom1').run('fin');

model.material.create('mat1', 'Common', 'comp1');
model.material('mat1').propertyGroup.create('Enu', 'Young''s modulus and Poisson''s ratio');

model.physics.create('actd', 'TransientPressureAcoustics', 'geom1');
model.physics('actd').create('mls1', 'TransientMonopoleLineSource', 0);
model.physics('actd').feature('mls1').selection.set([3]);



for i=1:size_of_x(2)
    domstring=strcat('pdom',num2str(i));
    model.probe.create(domstring, 'DomainPoint');
    model.probe(domstring).model('comp1');
end






model.view('view1').axis.set('abstractviewxscale', '0.0033112582750618458');
model.view('view1').axis.set('ymin', '-0.831125795841217');
model.view('view1').axis.set('xmax', '0.7682119011878967');
model.view('view1').axis.set('abstractviewyscale', '0.0033112585078924894');
model.view('view1').axis.set('abstractviewbratio', '-0.46523183584213257');
model.view('view1').axis.set('abstractviewtratio', '0.46523183584213257');
model.view('view1').axis.set('abstractviewrratio', '0.862582802772522');
model.view('view1').axis.set('xmin', '-0.7682119011878967');
model.view('view1').axis.set('abstractviewlratio', '-0.862582802772522');
model.view('view1').axis.set('ymax', '0.831125795841217');

model.material('mat1').label('Concrete');
model.material('mat1').set('family', 'concrete');
model.material('mat1').propertyGroup('def').set('thermalexpansioncoefficient', {'10e-6[1/K]' '0'
model.material('mat1').propertyGroup('def').set('density', '2300[kg/m^3]');
model.material('mat1').propertyGroup('def').set('thermalconductivity', {'1.8[W/(m*K)]' '0' '0' '
model.material('mat1').propertyGroup('def').set('heatcapacity', '880[J/(kg*K)]');


model.material('mat1').propertyGroup('def').set('soundspeed', c);
model.material('mat1').propertyGroup('Enu').set('youngsmodulus', '25e9[Pa]');
model.material('mat1').propertyGroup('Enu').set('poissonsratio', '0.33');

model.physics('actd').feature('mls1').set('Type', 'GaussianPulse');
model.physics('actd').feature('mls1').set('AperLength', '10');


model.physics('actd').feature('mls1').set('f0', f);
model.physics('actd').feature('mls1').set('tp', '0.0012[s]');

model.mesh('mesh1').run;



for i=1:size_of_x(2)
```

```
    domstring=strcat('pdom',num2str(i));
    pstring=strcat('ppb',num2str(i));
    x_string=num2str(x(1,i));
    y_string=num2str(x(2,i));
    model.probe(domstring).setIndex('coords2', x_string, 0, 0);
    model.probe(domstring).setIndex('coords2', y_string, 0, 1);
    model.probe(domstring).feature(pstring).set('window', 'window1');

end




model.study.create('std1');
model.study('std1').create('time', 'Transient');

model.sol.create('sol1');
model.sol('sol1').study('std1');
model.sol('sol1').attach('std1');
model.sol('sol1').create('st1', 'StudyStep');
model.sol('sol1').create('v1', 'Variables');
model.sol('sol1').create('t1', 'Time');
model.sol('sol1').feature('t1').create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature.remove('fcDef');

model.study('std1').feature('time').set('initstudyhide', 'on');
model.study('std1').feature('time').set('initsolhide', 'on');
model.study('std1').feature('time').set('solnumhide', 'on');
model.study('std1').feature('time').set('notstudyhide', 'on');
model.study('std1').feature('time').set('notsolhide', 'on');
model.study('std1').feature('time').set('notsolnumhide', 'on');



for i=2:size_of_x(2)
    solstring=strcat('sol',num2str(i));
    model.sol.create(solstring);
    model.sol(solstring).study('std1');
end



for i=1:size_of_x(2)
    datastring=strcat('dset',num2str(i));
    model.result.dataset.remove(datastring);
end



for i=1:size_of_x(2)
    domstring=strcat('pdom',num2str(i));
    model.probe(domstring).genResult([]);
end



model.study('std1').feature('time').set('tlist', 'range(0,0.0001,0.002)');
model.sol('sol1').attach('std1');
model.sol('sol1').feature('t1').set('timemethod', 'genalpha');
model.sol('sol1').feature('t1').set('tlist', 'range(0,0.0001,0.002)');
model.sol('sol1').runAll;



for i=2:size_of_x(2)
    solstring=strcat('sol',num2str(i));
    parstring=strcat('Parametric Solutions',num2str(i-1));
    model.sol(solstring).label(parstring);
end



model.result.dataset.remove('dset1');
model.result.remove('pg1');
model.result.dataset.create('dset1', 'Solution');
model.result.dataset('dset1').set('solution', 'sol1');
model.result.create('pg1', 'PlotGroup2D');
model.result('pg1').label('Acoustic Pressure (actd)');
model.result('pg1').set('oldanalysistype', 'noneavailable');
model.result('pg1').set('solvertype', 'none');
model.result('pg1').set('solnum', 1);
model.result('pg1').set('showlooplevel', {'off' 'off' 'off'});
model.result('pg1').set('oldanalysistype', 'noneavailable');
model.result('pg1').set('data', 'dset1');
model.result('pg1').feature.create('surf1', 'Surface');
model.result('pg1').feature('surf1').set('oldanalysistype', 'noneavailable');
model.result('pg1').feature('surf1').set('solvertype', 'none');
model.result('pg1').feature('surf1').set('data', 'parent');



for i=1:size_of_x(2)
    domstring=strcat('pdom',num2str(i));
    model.probe(domstring).genResult('none');
end



model.sol('sol1').runAll;

model.result('pg1').run;

result_path=strcat(pwd,'\probedata.csv');

model.result.table('tbl1').save(result_path);

%use following two commands to get .mph (regular comsol) file of system
%model_path=strcat(pwd,'\physicalmodel.mph');
%model.save(model_path)
```

4.3 getz.m

```
function M  = getz(n,parameters,x)
%gets the sensor data differences at point n from comsol

k=size(x);%number of sensors
```

```
veclength=k(2);
pressure_treshold=parameters(3);

comsol_model_improved1(n,parameters,x);%run comsol to build csv file


%import data from this csv
M=zeros(veclength,1);
csv_string='probedata.csv';
datamatrix=csvread(csv_string,5);
for kk=1:veclength
   datacolumn=datamatrix(:,kk+1);%1st colmn is time, others pressure
   M(kk)=datamatrix(find(datacolumn>pressure_treshold,1),1);

end

M=difz(M);

end
```

4.4 difz.m

```
function y=difz(x)
%computes all n(n-1)/2 differences from a nx1 vector

q=combnk(x,2);
y=q(:,1)-q(:,2);



end
```