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

3-dimensional version code documentation

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 of that report should be within this file repository, please read it first to gain insight of the subject, if you are not familiar with it.

This 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.

Contents

1 Introduction

As previously mentioned, 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

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

see revisition of report, located in this repository, for more information.

For non-convex elements the minimum-energy-metric is not easily calculable. This text's solution is to use a search method to solve this inversion problem, now in full three dimension. The solution is tried with Gauss-Newton method and a grid search method.

2 Locating Acoustic Emission Source, Fracture

This algorithm three-dimensionalized the original algorithm, see reporsitory text. The algorithm transforms data, i.e times when a sensor registers larger than threshold value pressure wave, to differences of these times. For the fracture location algorithm, the data is set to time-differences

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

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

$$h(x^0)_i = (D(x^0, x^{A_{i1}}) - D(x^0, x^{A_{i2}}))/c, \quad || \quad i \in \{1, ..., N(N-1)/2\}, \tag{3}$$

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

$$\frac{1}{\rho c^2}\frac{\partial^2 p}{\partial t^2} - \nabla \cdot \left(\frac{\nabla p}{\rho}\right) = Q_m, \tag{4}$$

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

$$Q_m = \frac{\partial Q_s}{\partial t}\delta(x - x^0), \tag{5}$$

where

$$Q_s = ae^{-\pi^2 f_0^2 (t-t_p)^2} \tag{6}$$

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

$$||z - h(x^0)||^2. \tag{7}$$

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

$$J(x^0) = \left(\frac{\partial h(x^0)}{\partial x_1^0} \quad \frac{\partial h(x^0)}{\partial x_2^0} ...\right). \tag{8}$$

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

$$\frac{\partial h(x^0)}{\partial x_j^0} = (h(x^0 + \epsilon e_j) - h(x^0))/\epsilon \quad || \quad j \in \{1, 2, ...\}, \tag{9}$$

where 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

$$x^{0,k+1} = x^{0,k} + \alpha^k J(x^{0,k})^\dagger (z - h(x^{0,k})) \tag{10}$$

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.

In grid search, we try different points and pick the most suitable point that minimizes functional (7).

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 dependted 3D 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 cuboid with one cubicmeter area is drawn. A cylinder is drawn inside this cuboid. This cylinder is differenciated from the cuboid. 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 cylinderical 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. If one picks too few probes, the saved **.m** file of the model does not show necessary amount of COMSOL's own logic for naming these probes by user. We need this logic to automate probe naming with MATLAB's loop command **for**. 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_model3d.m**.

3.1 Preparing comsol_model3d.m

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

```
function comsol_model3d(fault_location,parameters,sensor_locations)
```

and last line

```
out=model
```

changer to ending command

```
end
```

The file should have lines like these:

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

```
model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 0, 0);
model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 1, 0);
model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 2, 0);
```

This list of lines creates our sourcepoint and sets it into location (-0.3,-0.3,-0.3) and runs the geometry again. Let us change this to accept our own parameters

instead of constant point. See that the parameters, like **fault_x** are strings, with **num2str()** command.

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

```
% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 0, 0);
% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 1, 0);
% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 2, 0);

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').feature('pt1').setIndex('p', fault_z, 2, 0);
```

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));
        z_string=num2str(x(3,i));
        model.probe(domstring).setIndex('coords2', x_string, 0, 0);
        model.probe(domstring).setIndex('coords2', y_string, 0, 1);
        model.probe(domstring).setIndex('coords2', z_string, 0, 2);
        model.probe(domstring).feature(pstring).set('window', 'window1');

    end
```

Please note that MATLAB uses numbering (1,2,3...) where COMSOL (Java-built) uses (0,1,2...). Previous 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 and main_gridsearch.m

The **main.m** is, as it's name says, main script file. This time it runs only the Gauss-Newton algorithm and grid search algorithm is ran by separate script named **main_gridsearch.m** It builds necessary mathematics of the algorithm and calls **comsol_model3d.m** script as it's servant. The program's hierarchy is described in another file in this repository. When **main.m** calls **comsol_model3d.m**, a data file, **probetable.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. The visualisation is made by a script, but there might be a ways to use same model file in both COMSOL and MATLAB. 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 or grid search algorithm. On Gauss-Newton algorithm, it is important to have big enough epsilon for perturbation, as to change the mesh element. Too small perturbation gives same result and the algorithm halts.

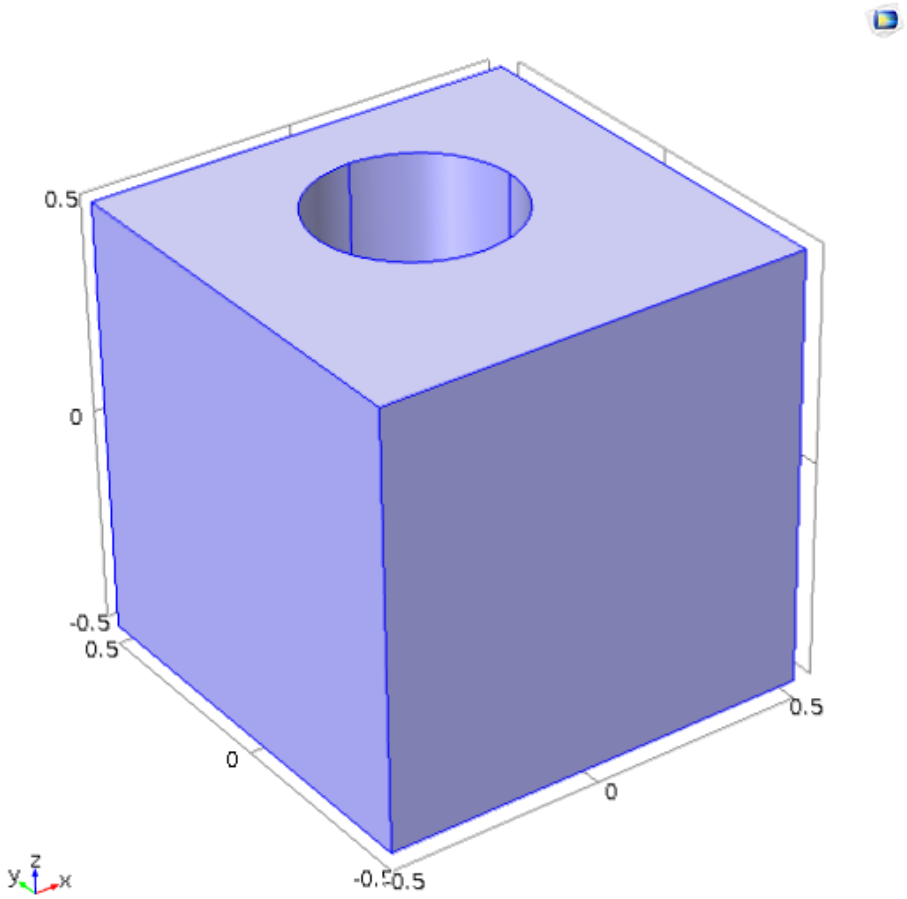

Figure 1: Basic COMSOL model.

4 Results and Discussion

The both algorithms are tested with fault location (-0.3, -0.3, -0.3). Gauss-Newton algorithm's initial startpoint is (0.35, 0.35, 0.35), which should lie well behind the void at opposite corner to represent a worst-case-scenario. Figure 2 shows one **main_gridsearch.m** output with twelve sensors. Figures 3 and 4 shows one of **main.m** output with twelve sensors. These results from algorithms seems to be near the original fault location. Building a real physical model and testing it with real sensors and using these or similar algorithms could be an intresting field of study and eliminate all possibilities of inverse crime. The solution time was somewhat lengthy with my Intel®i5-4690k processor, but it could be acceptable on structures with fault frequency less than one per 5 minutes. All script files are after reference list as an appendix.

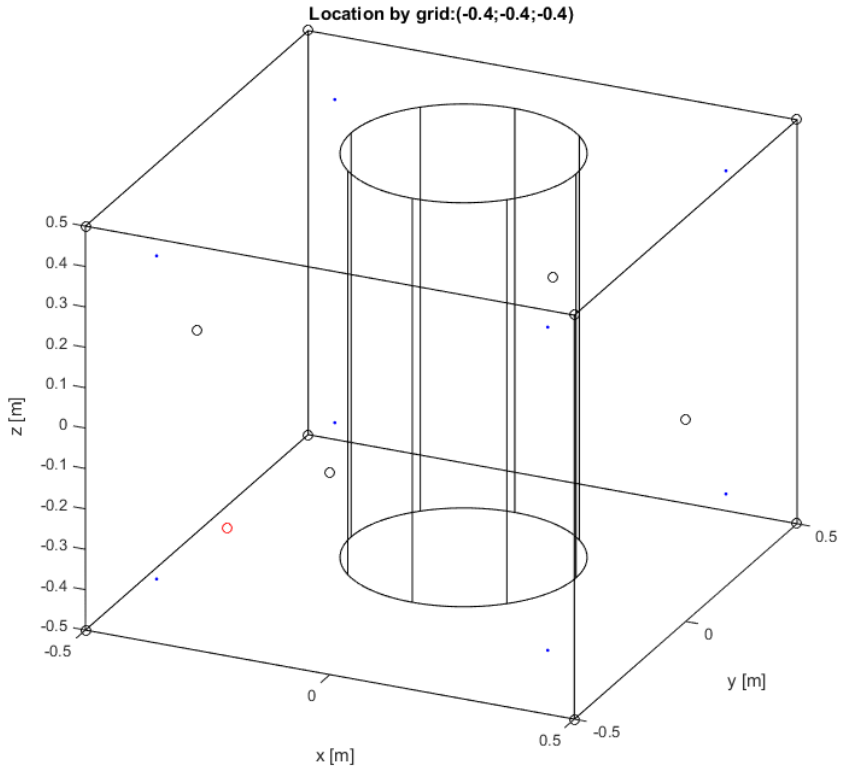

Figure 2: MATLAB output of main_gridsearch.m with grid constant 2. The true location is coordinate (-0.3, 0.3, 0.3), so the algorithm found closest point of the $2^3 = 8$ grid points.

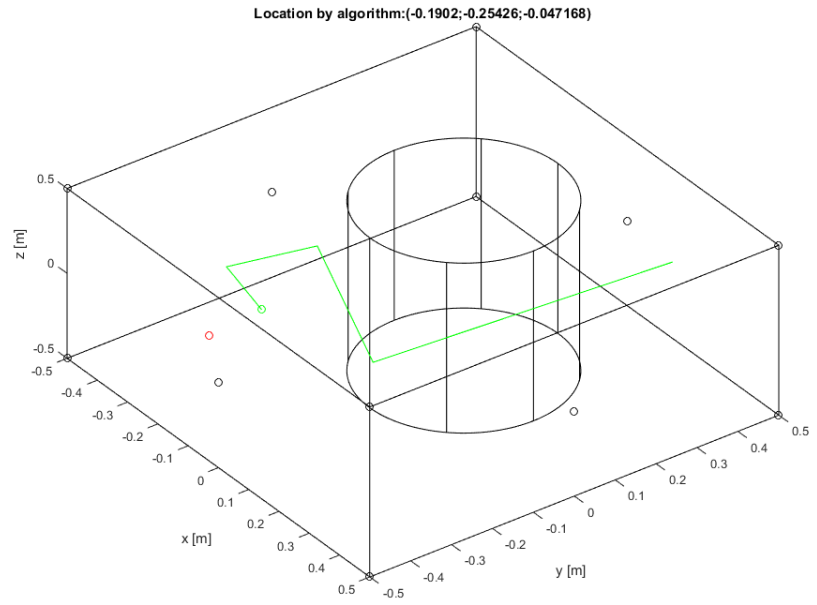

Figure 3: MATLAB output of main.m with 4 iterations. The true location is coordinate (-0.3, 0.3, 0.3), so the algorithm found a close solution.

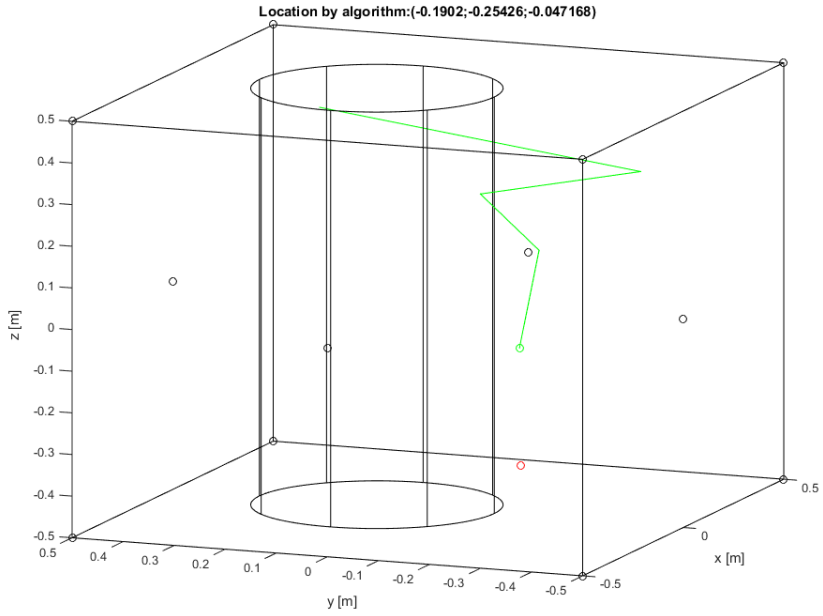

Figure 4: MATLAB output of main.m with 4 iterations. The true location is coordinate (-0.3, 0.3, 0.3), so the algorithm found a close solution. Another view.

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).

5 Program files

5.1 main.m

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

is_drawnow=1;%1=draw figure during algorithm 0=off

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

x=[0.5  -0.5     0     0    0.5     0.5    0.5     0.5   -0.5  -0.5  -0.5
-0.5;%x
       0     0   0.5  -0.5    0.5    -0.5    0.5    -0.5    0.5  -0.5
0.5   -0.5;%y
       0     0     0     0    0.5     0.5   -0.5    -0.5    0.5   0.5  -0.5
-0.5];%z
   %   1     2     3     4      5       6      7       8      9    10
11


%vertices of rectangular cuboid, 16 to draw
element_vertices_todraw=0.5*[+1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 -1 -1 -1 -1;%x
                             -1 -1 -1 -1 -1 +1 +1 +1 +1 +1 +1 -1 -1 +1 +1 -1;%y
                             -1 +1 +1 -1 -1 -1 +1 +1 -1 -1 +1 +1 +1 +1 -1 -1];%z
                          %   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15



%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;];
```

```
%iterations
k=4;



mesh_size=7;% mesh size in data(1=fine...9=extremely coarse [please: max 7])
%freq in data
f=25000;
%pulse speed
c=3200;

pressure_threshold=0.5e8;%pressure where signal is registered
parameters=[f,c,pressure_threshold,mesh_size,hole_radius,hole_loc_x,hole_loc_y];



lambda=c/f;%wavelength


%true damage location


loc=[-0.3 -0.3 -0.3]';



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


%change mesh to another to prevent inverse crime
mesh_size=7;
%freq in inversion
f=25000;
parameters=[f,c,pressure_threshold,mesh_size,hole_radius,hole_loc_x,hole_loc_y];
%step lenght
alpha=1;
%perturbation epsilon=a*lambda+b
epsa=0;%1/6;%0.1
epsb=0.12;%0.125
eps=epsa*lambda+epsb;
%breaking constant
s=0.5;

history_of_iterations=zeros(3,k);



NN=size(x);%amoutn of sensors
NN=NN(2);
n=[0.35 0.35 0.35]';%initial guess to algorithm




fig1=figure;
axis([-0.5 0.5 -0.5 0.5 -0.5 0.5]);
```

```
figure(fig1);

%draw sensors
plot3(x(1,:),x(2,:),x(3,:),'o','Color','black');
hold on;
figure(fig1);
%draw element
plot3(element_vertices_todraw(1,:),element_vertices_todraw(2,:),element_vertices_todraw(3,:),'Colo
figure(fig1);
xlabel('x [m]');
ylabel('y [m]');
zlabel('z [m]');



t = linspace(0,2*pi);
zminus=-0.5*ones(size(t));
zplus=0.5*ones(size(t));

%draw void outer circles

figure(fig1);
plot3(hole_loc_x+hole_radius*cos(t),hole_loc_y+hole_radius*sin(t),zminus,'color','black');

figure(fig1);
plot3(hole_loc_x+hole_radius*cos(t),hole_loc_y+hole_radius*sin(t),zplus,'color','black');


lineamount=8;
t = linspace(0,2*pi,lineamount+1);
tsize=size(t);

%draw lines circle to circle
for i=1:tsize(2)

    vrtx_todraw=zeros(3,2);
    vrtx_todraw(:,1)=[hole_loc_x+hole_radius*cos(t(i));hole_loc_y+hole_radius*sin(t(i));0.5];
    vrtx_todraw(:,2)=[hole_loc_x+hole_radius*cos(t(i));hole_loc_y+hole_radius*sin(t(i));-0.5];


    figure(fig1);
    plot3(vrtx_todraw(1,:),vrtx_todraw(2,:),vrtx_todraw(3,:),'color','black');


end







figure(fig1);
plot3(loc(1),loc(2),loc(3),'o','color','red');


if is_drawnow==1
    drawnow;
```

```
end




%initialize for algorithm
% h=zeros(NN,1);
% J1=zeros(NN,1);
% J2=zeros(NN,1);
% J3=zeros(NN,1);



%start algorithm

for index=1:k


%update h

h=getz(n,parameters,x);
%check if forward or backward difference due limitations
if n(1)>0
varx=-1;
end

if n(2)>0
vary=-1;
end

if n(3)>0
varz=-1;
end

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


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



%update J (jacobian)
J=[varx*(hx-h)/eps   vary*(hy-h)/eps  varz*(hz-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=4*eps*([rand rand rand]-0.5)';
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(1) ;n(2)]-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

for i=1:3
    if n(i)>0.5-eps/2
       n(i)=0.5-eps*2;%push further back
    end

    if n(i)<-0.5+eps/2
       n(i)=-0.5+eps*2;%push further back
    end


end









history_of_iterations(:,index)=n;
figure(fig1);
linematrix_todraw=[n lastn];
plot3(linematrix_todraw(1,:),linematrix_todraw(2,:),linematrix_todraw(3,:),'color','green');

if is_drawnow==1
    drawnow;
end


end
```

```
figure(fig1);
plot3(n(1),n(2),n(3),'o','color','green');
```

```
figure(fig1);
titlestr=strcat('Location by algorithm:(',num2str(n(1)),';', num2str(n(2)),';', num2str(n(3))
,')');
title(titlestr);
```

5.2 main_gridsearch.m

```
%calculates failure location in concrete structure with internal circ void
%uses grid search

%grid constant for search, dot amount amount is grid^3
gridc=2;


is_drawnow=1;%1=draw figure during algorithm 0=off

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

x=[0.5  -0.5     0     0    0.5     0.5    0.5     0.5   -0.5  -0.5  -0.5
-0.5;%x
       0     0   0.5  -0.5    0.5    -0.5    0.5    -0.5    0.5  -0.5
0.5   -0.5;%y
       0     0     0     0    0.5     0.5   -0.5    -0.5    0.5   0.5
-0.5   -0.5];%z
    %   1     2     3     4      5       6      7       8      9    10
11


%vertices of rectangular cuboid, 16 to draw
element_vertices_todraw=0.5*[+1 +1 -1 -1 +1 +1 +1 -1 -1 +1 +1 +1 -1 -1 -1 -1;%x
                             -1 -1 -1 -1 -1 +1 +1 +1 +1 +1 +1 -1 -1 +1 +1 -1;%y
                             -1 +1 +1 -1 -1 -1 +1 +1 -1 -1 +1 +1 +1 +1 -1 -1];%z
                          %   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15


%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;];


mesh_size=8;% mesh size in data(1=fine...9=extremely coarse [please: max 8])
%freq in data
f=25000;
```

```
%pulse speed
c=3200;

pressure_threshold=0.5e8;%pressure where signal is registered
parameters=[f,c,pressure_threshold,mesh_size,hole_radius,hole_loc_x,hole_loc_y];



lambda=c/f;%wavelength


%true damage location


loc=[-0.3 -0.3 -0.3]';

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


mesh_size=8;% mesh size in data(1=fine...9=extremely coarse)
%freq in inversion
f=25000;
parameters=[f,c,pressure_threshold,mesh_size,hole_radius,hole_loc_x,hole_loc_y];







fig1=figure;
axis([-0.5 0.5 -0.5 0.5 -0.5 0.5]);


figure(fig1);

%draw sensors
plot3(x(1,:),x(2,:),x(3,:),'o','Color','black');
hold on;
figure(fig1);
%draw element
plot3(element_vertices_todraw(1,:),element_vertices_todraw(2,:),element_vertices_todraw(3,:),'Co
figure(fig1);
xlabel('x [m]');
ylabel('y [m]');
zlabel('z [m]');



t = linspace(0,2*pi);
zminus=-0.5*ones(size(t));
zplus=0.5*ones(size(t));

%draw void outer circles

figure(fig1);
plot3(hole_loc_x+hole_radius*cos(t),hole_loc_y+hole_radius*sin(t),zminus,'color','black');

figure(fig1);
plot3(hole_loc_x+hole_radius*cos(t),hole_loc_y+hole_radius*sin(t),zplus,'color','black');
```

```
lineamount=8;
t = linspace(0,2*pi,lineamount+1);
tsize=size(t);

%draw lines circle to circle
for i=1:tsize(2)

    vrtx_todraw=zeros(3,2);
    vrtx_todraw(:,1)=[hole_loc_x+hole_radius*cos(t(i));hole_loc_y+hole_radius*sin(t(i));0.5];
    vrtx_todraw(:,2)=[hole_loc_x+hole_radius*cos(t(i));hole_loc_y+hole_radius*sin(t(i));-0.5];


    figure(fig1);
    plot3(vrtx_todraw(1,:),vrtx_todraw(2,:),vrtx_todraw(3,:),'color','black');


end









figure(fig1);
plot3(loc(1),loc(2),loc(3),'o','color','red');


if is_drawnow==1
    drawnow;
end
```

```
A=linspace(-0.4,0.4,gridc);
B=linspace(-0.4,0.4,gridc);
C=linspace(-0.4,0.4,gridc);

[A,B,C]=meshgrid(A,B,C);
 D=zeros(gridc,gridc,gridc);
```

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

            if (norm([nn1(1) nn1(2)]'-hole_loc_vec)<hole_radius )
                D(ix,iy)=NaN;%NaN if inside void
            else

                figure(fig1);


                plot3(nn1(1),nn1(2),nn1(3),'.','color','blue')
                drawnow
                hold on


                h=getz(nn1,parameters,x);
                D(ix,iy,iz)=norm(z-h);

            end
        end
    end
end

x_val=num2str(A(D==min(min(min(D)))));
y_val=num2str(B(D==min(min(min(D)))));
z_val=num2str(C(D==min(min(min(D)))));


titlestr=strcat('Location by grid:(',x_val,';',y_val,';',z_val,')');
title(titlestr);
```

5.3 comsol_model3d.m

```
function comsol_model3d(location,parameters,sensors)


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

mesh_size = parameters(4); % mesh size(1=fine...9=extremely coarse)

%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(location(1));
fault_y=num2str(location(2));
fault_z=num2str(location(3));

matlab_path_string=pwd;
```

```
%
% comsol_model3d.m
%
% Model exported on Dec 27 2020, 12:33 by COMSOL 5.1.0.234.

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

model = ModelUtil.create('Model');

model.modelPath('C:\Users\Mika\OneDrive for Business\concrete_3d');

model.label('comsol_model3d.mph');

model.comments(['Untitled\n\n']);

model.modelNode.create('comp1');

model.file.clear;

model.geom.create('geom1', 3);

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

model.geom('geom1').create('blk1', 'Block');
model.geom('geom1').feature('blk1').set('pos', {'-0.5' '-0.5' '-0.5'});

model.geom('geom1').create('cyl1', 'Cylinder');


% model.geom('geom1').feature('cyl1').set('r', '0.3');

model.geom('geom1').feature('cyl1').set('r', hole_radius);

% model.geom('geom1').feature('cyl1').set('pos', {'0' '0' '-0.5'});
model.geom('geom1').feature('cyl1').set('pos', {hole_loc_x hole_loc_y '-0.5'});




model.geom('geom1').create('dif1', 'Difference');
model.geom('geom1').feature('dif1').selection('input2').set({'cyl1'});
model.geom('geom1').feature('dif1').selection('input').set({'blk1'});



model.geom('geom1').create('pt1', 'Point');
model.geom('geom1').feature('pt1').label('Sourcepoint');



% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 0, 0);
% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 1, 0);
% model.geom('geom1').feature('pt1').setIndex('p', '-0.3', 2, 0);

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').feature('pt1').setIndex('p', fault_z, 2, 0);


model.geom('geom1').run;
model.geom('geom1').run('fin');

model.material.create('mat2', 'Common', 'comp1');
model.material('mat2').propertyGroup.create('Enu', 'Young''s modulus and Poisson''s ratio');

model.physics.create('actd', 'TransientPressureAcoustics', 'geom1');
model.physics('actd').create('mps1', 'TransientMonopolePointSource', 0);
model.physics('actd').feature('mps1').selection.set([5]);

model.mesh('mesh1').create('ftet1', 'FreeTet');

model.result.table.create('tbl1', 'Table');

% model.probe.create('pdom1', 'DomainPoint');
% model.probe.create('pdom2', 'DomainPoint');
% model.probe.create('pdom3', 'DomainPoint');
% model.probe.create('pdom4', 'DomainPoint');
% model.probe('pdom1').model('comp1');
% model.probe('pdom2').model('comp1');
% model.probe('pdom3').model('comp1');
% model.probe('pdom4').model('comp1');

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



model.material('mat2').label('Concrete');
model.material('mat2').set('family', 'concrete');
model.material('mat2').propertyGroup('def').set('thermalexpansioncoefficient', {'10e-6[1/K]' '0'
model.material('mat2').propertyGroup('def').set('density', '2300[kg/m^3]');
model.material('mat2').propertyGroup('def').set('thermalconductivity', {'1.8[W/(m*K)]' '0' '0' '0
model.material('mat2').propertyGroup('def').set('heatcapacity', '880[J/(kg*K)]');



%model.material('mat2').propertyGroup('def').set('soundspeed', '3200');

model.material('mat2').propertyGroup('def').set('soundspeed', c);


model.material('mat2').propertyGroup('Enu').set('youngsmodulus', '25e9[Pa]');
model.material('mat2').propertyGroup('Enu').set('poissonsratio', '0.33');

model.physics('actd').feature('mps1').set('Type', 'GaussianPulse');
model.physics('actd').feature('mps1').set('A', '10');

% model.physics('actd').feature('mps1').set('f0', '25000[Hz]');

model.physics('actd').feature('mps1').set('f0', f);


model.physics('actd').feature('mps1').set('tp', '0.0012[s]');
```

```
%model.mesh('mesh1').feature('size').set('hauto', 7);
model.mesh('mesh1').feature('size').set('hauto', mesh_size);
model.mesh('mesh1').run;

model.result.table('tbl1').label('Probe Table 1');

% model.probe('pdom1').setIndex('coords3', '0.5', 0, 0);
% model.probe('pdom1').setIndex('coords3', '0', 0, 1);
% model.probe('pdom1').setIndex('coords3', '0', 0, 2);
% model.probe('pdom1').feature('ppb1').set('window', 'window1');
% model.probe('pdom1').feature('ppb1').set('table', 'tbl1');
% model.probe('pdom2').setIndex('coords3', '-0.5', 0, 0);
% model.probe('pdom2').setIndex('coords3', '0', 0, 1);
% model.probe('pdom2').setIndex('coords3', '0', 0, 2);
% model.probe('pdom2').feature('ppb2').set('window', 'window1');
% model.probe('pdom2').feature('ppb2').set('table', 'tbl1');
% model.probe('pdom3').setIndex('coords3', '0', 0, 0);
% model.probe('pdom3').setIndex('coords3', '0.5', 0, 1);
% model.probe('pdom3').setIndex('coords3', '0', 0, 2);
% model.probe('pdom3').feature('ppb3').set('window', 'window1');
% model.probe('pdom3').feature('ppb3').set('table', 'tbl1');
% model.probe('pdom4').setIndex('coords3', '0', 0, 0);
% model.probe('pdom4').setIndex('coords3', '-0.5', 0, 1);
% model.probe('pdom4').setIndex('coords3', '0', 0, 2);
% model.probe('pdom4').feature('ppb4').set('window', 'window1');
% model.probe('pdom4').feature('ppb4').set('table', 'tbl1');

for i=1:sensor_amount
    probe_name_string=strcat('pdom',num2str(i));
    probepoint_name_string=strcat('ppb',num2str(i));
    for j=0:2%direction 0=x 1=y 2=z java way
        coordinate_string=num2str(sensors(j+1,i));%plus one to matlab way
        model.probe(probe_name_string).setIndex('coords3', coordinate_string, 0, j);
    end

    model.probe(probe_name_string).feature(probepoint_name_string).set('window', 'window1');
    model.probe(probe_name_string).feature(probepoint_name_string).set('table', 'tbl1');
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');
```

```
model.result.dataset.create('dset2', 'Solution');


% model.result.dataset.create('cpt1', 'CutPoint3D');
% model.result.dataset.create('cpt2', 'CutPoint3D');
% model.result.dataset.create('cpt3', 'CutPoint3D');
% model.result.dataset.create('cpt4', 'CutPoint3D');

for i=1:sensor_amount
    cutpointname_string=strcat('cpt',num2str(i));
    model.result.dataset.create(cutpointname_string, 'CutPoint3D');
end



%model.result.dataset('dset2').set('probetag', 'pdom4');
%force first probe to main probe
model.result.dataset('dset2').set('probetag', 'pdom1');



% model.result.dataset('cpt1').set('probetag', 'pdom4');
% model.result.dataset('cpt1').set('data', 'dset2');
% model.result.dataset('cpt2').set('probetag', 'pdom1');
% model.result.dataset('cpt2').set('data', 'dset2');
% model.result.dataset('cpt3').set('probetag', 'pdom2');
% model.result.dataset('cpt3').set('data', 'dset2');
% model.result.dataset('cpt4').set('probetag', 'pdom3');
% model.result.dataset('cpt4').set('data', 'dset2');

%COMSOL set last probe to main
%let us force first to first and force logic cptX->pdomX
for i=1:sensor_amount
    cutpoint_string=strcat('cpt',num2str(i));
    probe_name_string=strcat('pdom',num2str(i));

    model.result.dataset(cutpoint_string).set('probetag', probe_name_string);
    model.result.dataset(cutpoint_string).set('data', 'dset2');

end



% model.result.numerical.create('pev1', 'EvalPoint');
% model.result.numerical.create('pev2', 'EvalPoint');
% model.result.numerical.create('pev3', 'EvalPoint');
% model.result.numerical.create('pev4', 'EvalPoint');
%
%
%
% model.result.numerical('pev1').set('probetag', 'pdom4/ppb4');
% model.result.numerical('pev2').set('probetag', 'pdom1/ppb1');
% model.result.numerical('pev3').set('probetag', 'pdom2/ppb2');
% model.result.numerical('pev4').set('probetag', 'pdom3/ppb3');
%


for i=1:sensor_amount
    evalpoint_string=strcat('pev',num2str(i));
    probetag_string=strcat('pdom',num2str(i),'/ppb',num2str(i));
```

```
    model.result.numerical.create(evalpoint_string, 'EvalPoint');
    model.result.numerical(evalpoint_string).set('probetag', probetag_string);
end



model.result.create('pg1', 'PlotGroup3D');
model.result.create('pg2', 'PlotGroup3D');
model.result.create('pg3', 'PlotGroup1D');




model.result('pg1').create('surf1', 'Surface');
model.result('pg2').create('iso1', 'Isosurface');
model.result('pg3').set('probetag', 'window1_default');
model.result('pg3').create('tblp1', 'Table');



%model.result('pg3').feature('tblp1').set('probetag', 'pdom4/ppb4,pdom1/ppb1,pdom2/ppb2,pdom3/ppb3



probetag_string_all='pdom1/ppb1';
for i=2:sensor_amount%start from second
    additive=strcat(',pdom',num2str(i),'/ppb',num2str(i));
    probetag_string_all=strcat(probetag_string_all,additive);
end

model.result('pg3').feature('tblp1').set('probetag', probetag_string_all);

% model.probe('pdom1').genResult([]);
% model.probe('pdom2').genResult([]);
% model.probe('pdom3').genResult([]);
% model.probe('pdom4').genResult([]);


for i=1:sensor_amount
    probe_name_string=strcat('pdom',num2str(i));
    model.probe(probe_name_string).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('tlist', 'range(0,0.0001,0.002)');
model.sol('sol1').feature('t1').set('timemethod', 'genalpha');
model.sol('sol1').runAll;

model.result.dataset('dset2').label('Probe Solution 2');
model.result.dataset('dset2').set('frametype', 'spatial');


% model.result.dataset('cpt1').set('data', 'dset2');
% model.result.dataset('cpt2').set('data', 'dset2');
% model.result.dataset('cpt3').set('data', 'dset2');
% model.result.dataset('cpt4').set('data', 'dset2');
```

```
for i=1:sensor_amount
    cutpoint_name_string=strcat('cpt',num2str(i));
    model.result.dataset(cutpoint_name_string).set('data', 'dset2');
end


model.result.dataset.remove('dset3');



model.result.numerical('pev1').setResult;



% model.result.numerical('pev2').appendResult;
% model.result.numerical('pev3').appendResult;
% model.result.numerical('pev4').appendResult;

for i=2:sensor_amount%start with 2
    evalpoint_name_string=strcat('pev',num2str(i));
    model.result.numerical(evalpoint_name_string).appendResult;
end



model.result('pg1').label('Acoustic Pressure (actd)');
model.result('pg1').set('looplevel', {'18'});
model.result('pg2').label('Acoustic Pressure, Isosurfaces (actd)');
model.result('pg2').set('looplevel', {'1'});
model.result('pg2').feature('iso1').set('number', '10');
model.result('pg3').label('Probe Plot Group 3');
model.result('pg3').set('xlabel', 't');
model.result('pg3').set('windowtitle', 'Probe Plot 1');
model.result('pg3').set('xlabelactive', false);
model.result('pg3').feature('tblp1').label('Probe Table Graph 1');

model.sol('sol1').study('std1');

model.study('std1').feature('time').set('notlistsolnum', 1);
model.study('std1').feature('time').set('notsolnum', '1');
model.study('std1').feature('time').set('listsolnum', 1);
model.study('std1').feature('time').set('solnum', '1');

model.sol('sol1').feature.remove('t1');
model.sol('sol1').feature.remove('v1');
model.sol('sol1').feature.remove('st1');
model.sol('sol1').create('st1', 'StudyStep');
model.sol('sol1').feature('st1').set('study', 'std1');
model.sol('sol1').feature('st1').set('studystep', 'time');
model.sol('sol1').create('v1', 'Variables');
model.sol('sol1').feature('v1').set('control', 'time');
model.sol('sol1').create('t1', 'Time');
model.sol('sol1').feature('t1').set('tlist', 'range(0,0.0001,0.002)');
model.sol('sol1').feature('t1').set('plot', 'off');
model.sol('sol1').feature('t1').set('plotgroup', 'pg1');
model.sol('sol1').feature('t1').set('plotfreq', 'tout');
model.sol('sol1').feature('t1').set('probesel', 'all');
```

```
% model.sol('sol1').feature('t1').set('probes', {'pdom1' 'pdom2' 'pdom3' 'pdom4'});
probename_cell=cell(1,sensor_amount);%build cell on probe names
for i=1:sensor_amount
    probename_string=strcat('pdom',num2str(i));
    probename_cell{i}=probename_string;
end

model.sol('sol1').feature('t1').set('probes', probename_cell);


model.sol('sol1').feature('t1').set('probefreq', 'tsteps');
model.sol('sol1').feature('t1').set('rtol', 0.01);
model.sol('sol1').feature('t1').set('atolglobalmethod', 'scaled');
model.sol('sol1').feature('t1').set('atolglobal', 0.001);
model.sol('sol1').feature('t1').set('timemethod', 'genalpha');
model.sol('sol1').feature('t1').set('rhoinf', 0.75);
model.sol('sol1').feature('t1').set('maxorder', 5);
model.sol('sol1').feature('t1').set('minorder', 1);
model.sol('sol1').feature('t1').set('control', 'time');
model.sol('sol1').feature('t1').create('fc1', 'FullyCoupled');
model.sol('sol1').feature('t1').feature('fc1').set('linsolver', 'dDef');
model.sol('sol1').feature('t1').feature.remove('fcDef');
model.sol('sol1').attach('std1');



% model.probe('pdom1').genResult('none');
% model.probe('pdom2').genResult('none');
% model.probe('pdom3').genResult('none');
% model.probe('pdom4').genResult('none');

for i=1:sensor_amount
    probename_string=strcat('pdom',num2str(i));
    model.probe(probename_string).genResult('none');
end


model.sol('sol1').runAll;

model.result('pg1').run;
%model.result.table('tbl1').save('C:\Users\Mika\OneDrive for Business\concrete_3d\probetable.csv')

%save data to csv
result_path=strcat(matlab_path_string,'\probetable.csv');
model.result.table('tbl1').save(result_path);

%use following two commands to get .mph (regular comsol) file of system
%model_path=strcat(matlab_path_string,'\physicalmodel.mph');
%model.save(model_path)
%comment out when unneeded

end
```

5.4 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_threshold=parameters(3);
```

```
comsol_model3d(n,parameters,x);%run comsol to build csv file


%import data from this csv
M=zeros(veclength,1);
csv_string='probetable.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_threshold,1),1);

end

M=difz(M);

end
```

5.5 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
```