CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Project work:

Acoustic streaming modeling

Developed for OpenFOAM-extended 3.2

Author:
Milad Setareh

Peer reviewed by:
Håkan Nilsson

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

December 4, 2016

Learning outcomes

The reader will learn:

- the theory of wave propagation into a fluid
- how to model acoustic streaming
- how to use the sonicLiquidFoam solver and develop it
- how to use the buoyantBoussinesqSimpleFoam solver and develop it

Chapter 1

Introduction

1.1 Introduction

This tutorial describes how to use the sonicLiquidFoam and buoyantBoussinesqSimpleFoam for acoustic streaming modeling. sonicLiquidFoam is a transient compressible solver for modeling the propagation of wave into a liquid and buoyantBoussinesqSimpleFoam is a steady–state solver for buoyant, turbulent flow of incompressible fluids.

In order to do this, firstly, the sonicLiquidFoam solver is modified to calculate some source terms, Then, they are used as source terms in buoyantBoussinesqSimpleFoam solver to observe acoustic streaming. Also, it is possible to study heat transfer by using buoyantBoussinesqSimpleFoam solver.

1.2 Mathematical model

Some methods has been presented for modeling of acoustic streaming: slip velocity, perturbation method and direct method. In this report, acoustic streaming is modeled by perturbation method. In this method, the variables in the Navier–Stokes and continuity equations are written as a expansion of equilibrium. The equilibrium value corresponds to zero order, represent constant in time and space, in the absence of acoustic excitation. Therefore, the variables are expanded as:

$$\begin{aligned} \rho &= \rho^0 + \rho^1 + \rho^2 \\ u &= u^0 + u^1 + u^2 \\ p &= p^0 + p^1 + p^2 \end{aligned} \tag{1.1}$$

where (0) refers to zero order, (1) to first–order and (2) to second–order The components of third and higher orders are neglected, since they are insignificant for describing the acoustic streaming effects. The first–order variables represent the damped propagation of the acoustic wave, providing analysis of the instantaneous acoustic flow, while the second–order variables describe the average acoustic streaming effects.

1.2.1 First order equation

The first order continuity, momentum and sate equations are the following, respectively:

$$\frac{\partial \rho^1}{\partial t} + \rho^0 \nabla \cdot \left(u^1\right) = 0 \tag{1.2}$$

$$\rho^0 \frac{\partial u^1}{\partial t} = -\nabla \cdot p^1 + \mu \nabla^2 u^1 \tag{1.3}$$

$$p^1 = c^2 \rho^1 \tag{1.4}$$

Where ρ^0 represents the equilibrium density, constant in time and space, ρ^1 is the first–order density, u^2 is the first –order velocity and C is sound speed into domain. After solving the firstâorder system for all the required time steps, it is calculated a time average driving force. The time average value is obtained by integration of the driving force term during one wave period, after the stabilization of the periodic solution, and posterior division by the period duration. This time average driving force, described by Nyborg [1] can be applied as a source term to solve the second–order system.

1.2.2 Second order equation

The second–order Navier Stokes system considers the first–order solution as known data. It includes the second–order terms and describes the mass and body force sources. The time average values determined above allow analyzing the acoustic streaming effects in a large time scale. Therefore, solving the secondâorder system it is determined the mean global flow. The source term momentum and continuity equations are described by the following equations, where the term into brackets in equation represents the nonlinear driving force terms: (source terms for continuity equations is very low, So we can ignore it)

$$\frac{\partial \rho^2}{\partial t} + \rho^0 \nabla \cdot \left(u^2\right) = \left\langle -\nabla \cdot \left(\rho^1 u^1\right) \right\rangle \tag{1.5}$$

$$\rho^0 \frac{\partial u^2}{\partial t} = -\nabla \cdot p^2 + \mu \nabla^2 u^2 + \left\langle -\rho^1 \frac{\partial u^1}{\partial t} - \rho^0 \left(u^1 \cdot \nabla\right) u^1 \right\rangle \tag{1.6}$$

$$p^2 = c^2 \rho^2 \tag{1.7}$$

1.3 Pre-processing

This section covers the necessary setup needed to modify sonicLiquidFoam and buoyantBoussinesqSimpleFoam.

1.3.1 Getting started

We will now start to develop our new sonicLiquidFoam and buoyantBoussinesqSimpleFoam solvers in the local solvers directory. Copy the original solvers there, change the corresponding file names, and then compile the solver so that we make sure the basic setup is correct:

```
cd $WM_PROJECT_USER_DIR
mkdir -p applications/solvers/compressible/sonicLiquidFoam
cd applications/solvers/compressible
cp -r $FOAM_SOLVERS/compressible/sonicLiquidFoam .
cd sonicLiquidFoam
mv sonicLiquidFoam.C mySonicLiquidFoam.C
sed -i 's/sonicLiquidFoam/mySonicLiquidFoam/g' Make/files
sed -i 's/FOAM_APPBIN/FOAM_USER_APPBIN/g' Make/files
wclean
wmake
```

```
cd $WM_PROJECT_USER_DIR
mkdir -p applications/solvers/heatTransfer/buoyantBoussinesqSimpleFoam
cd applications/solvers/heatTransfer
cp -r $FOAM_SOLVERS/heatTransfer/buoyantBoussinesqSimpleFoam .
cd buoyantBoussinesqSimpleFoam
mv buoyantBoussinesqSimpleFoam.C myBuoyantBoussinesqSimpleFoam.C
sed -i 's/buoyantBoussinesqSimpleFoam/myBuoyantBoussinesqSimpleFoam/g' Make/files
sed -i 's/FOAM_APPBIN/FOAM_USER_APPBIN/g' Make/files
wclean
wmake
```

Once it compiles successfully, we can then start to modify the solvers for our own sake. First, we modify the mySonicLiquidFoam solver. So, we should go to that directory and do all things as below. to create source term fields, we should add below lines in the `creatFields.H` file.

```
    volVectorField sumAcousticForce
    (
        IOobject
        (
            "sumAcousticForce",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedVector("meanAcousticForce",  dimensionSet(0,1,-2,0,0,0,0),
        vector(0,0,0))
    );
```

```
volVectorField meanAcousticForce
    (
        IOobject
        (
            "meanAcousticForce",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh,
         dimensionedVector("meanAcousticForce", dimensionSet(0,1,-2,0,0,0,0),
         vector(0,0,0))
    );
```

Also, we change the initial value of ρ to `1+psi*p` in the `createFields.H`. For considering kinematic viscosity, it is better to change the word `mu` in `readTransportProperties.H` file to `nu`.

Now, we can modify the main mySonicLiquidFoam.C file to implement the first order governing equation. First, we declare a `float` variable named `iter` before `run.Time()` looping and initialize it with zero. Then, we change compressibleCourantNo.H to CourantNo.H everywhere in source code by the following line:

```
sed -i 's/compressibleCourantNo.H/CourantNo.H/g' mySonicLiquidFoam.C
```

the incompressible momentum equation is considered for first order equations,Therefore it does not need to solve density equation and we can remove it. for this purpose, delete the line: `#include "rhoEqn.H"`. it is located above UEqn in the source code. Now, we should implement first order equations. we can change momentum and pressure equations line by line, but I think it is better to remove all lines started `fvVectorMatrix UEqn` and ended with `rho = rho0 + psi*p`. Then, replace the below code instead of everything has been removed.

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  - fvm::laplacian(nu, U)
);

solve(UEqn == -fvc::grad(p));

// --- PISO loop

for (int corr=0; corr<nCorr; corr++)
{
    volScalarField rAU(1.0/UEqn.A());
    U = rAU*UEqn.H();

    surfaceScalarField phid
    (
        "phid",
        psi
       *(
            (fvc::interpolate(U) & mesh.Sf())
         // + fvc::ddtPhiCorr(rAU, rho, U, phi)
        )
    );

    phi = (1/psi)*phid;

    fvScalarMatrix pEqn
    (
        fvm::ddt(psi,p)
      + fvc::div(phi)
      //+ fvm::div(phid, p)
      - fvm::laplacian(rAU, p)
    );

    pEqn.solve();

    phi += pEqn.flux();
```

```
            #include "continuityErrs.H"

            U -= rAU*fvc::grad(p);
            U.correctBoundaryConditions();
        }
        rho = 1 + psi*p;
        iter=iter+1.0;
```

The most important part is calculation of driving force. The formula of the momentum driving force has been written in the previous sections. Now, we add the following lines at the end of `run.Time` loop.

```
sumAcousticForce=sumAcousticForce+fvc::div(phi, U)+(rho-1)*fvc::ddt(U);
meanAcousticForce=sumAcousticForce/iter;
```

The momentum driving force is calculated at the first line of the above box. Because, we need the its time average to import it in the second order equations, So at the second line of the above box, the driving force has been divided by the number of iterations. when the solution becomes converge, the `meanAcousticForce` variable expresses time averaging driving force for momentum equation.

Now, we have completed the implementation of all new features that the mySonicLiquidFoam solver should have. Type *wmake* in the terminal and compile the new solver. If everything is ok, we can go to second part which is the modification of the myBuoyantBoussinesqSimpleFoam solver.

Now, go to the myBuoyantBoussinesqSimpleFoam solver directory. these are some files and we just modify `createFields.H` and `UEqn.H` files.

it does nor need to do a lot of changes in these files, we just add some lines in these files, so that the solver can recognize the driving momentum source. First, we define the verb$meanAcousticForce$ in the `createFields.H` as follows:

```
volVectorField meanAcousticForce
   (
       IOobject
       (
           "meanAcousticForce",
           runTime.timeName(),
           mesh,
           IOobject::MUST_READ,
           IOobject::AUTO_WRITE
       ),
       mesh
   );
```

Now, it's time to add this source term to second order momentum equation. The formula of second order equations is written in the previous sections. As you can see, the convection term was ignored in that equations but for more accuracy, we consider it. we can ignore it just by removing the `fvm::div(phi, U)` in the `UEqn.H` file.

To import the driving momentum source into U equation, we should add `meanAcousticForce` in the `UEqn` as shown in the following:

```
    tmp<fvVectorMatrix> UEqn
    (
        fvm::div(phi, U)
      + turbulence->divDevReff(U)
      +meanAcousticForce
    );
```

Finally, compile the solver by typing `wmake`. Now we can use these two solvers to study acoustic streaming. the following report focuses on how to use these solvers and run a case.

Chapter 2

Test solvers and run a case

2.1 Domain and boundary condition

In this chapter, a simple case will be run and we will observe the results. To generate a wave in the boundary, I considered harmonic pressure boundary at one of the walls. The `groovyBC` has been used to implement that boundary condition. Note,`groovyBC` is a library in the swak4Foam package which has many types of boundary conditions.

2.2 Mesh generation and transportProperties

It is recommended to use cavity tutorial. Therefore, copy the cavity directory to run directory and change it in according to requirements. The below lines, copy the cavity tutorial to the local run directory.

```
mkdir -p $FOAM_RUN/cavity
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN/cavity
cd $FOAM_RUN/cavity
mv cavity cavityFirstOrder
```

Open blockMeshDict and create the mesh and set the boundary conditions as follows:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.0.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.0001;
```

```
vertices
(
    (0 0 -0.1)//0
    (10 0 -0.1)//1
    (10 20 -0.1)//2
    (0 20 -0.1)//3
    (0 0 0.1)//4
    (10 0 0.1)//5
    (10 20 0.1)//6
    (0 20 0.1)//7
    (14 0 -0.1)//8
    (14 20 -0.1)//9
    (14 0 0.1)//10
    (14 20 0.1)//11
    (24 0 -0.1)//12
    (24 20 -0.1)//13
    (24 0 0.1)//14
    (24 20 0.1)//15
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (400 700 1) simpleGrading (1 1 1)
    hex (1 8 9 2 5 10 11 6) (400 700 1) simpleGrading (1 1 1)
    hex (8 12 13 9 10 14 15 11) (400 700 1) simpleGrading (1 1 1)
);

boundary
(
    walls
    {
        type wall;
        faces
      (
        (0 4 7 3)
        (12 13 15 14)
      );
    }
    inlet_w1
    {
        type wall;
        faces
      (
        (3 2 6 7)
      );
    }

    acous
    {
        type wall;
        faces
      (
        (2 9 11 6)
      );
```

```
    }

    inlet_w2
    {
        type wall;
        faces
      (
        (9 13 15 11)
      );
    }
    outlet
    {
        type wall;
        faces
      (
            (0 1 5 4)
            (1 8 10 5)
            (8 12 14 10)
      );
    }
    frontAndBack
    {
        type empty;
        faces
      (
            (0 3 2 1)
            (1 2 9 8)
            (8 9 13 12)
            (4 7 6 5)
            (5 6 11 10)
            (10 11 15 14)
      );
    }
);
```

Now, we remove the `transportProperties` in the `cavity` and copy `transportProperties` from one of the tutorial cases in the sonicLiquidFoam directory as follows:

```
rm -rf $FOAM_RUN/cavity/cavityFirstOrder/constant/transportProperties
cp -r $FOAM_TUTORIALS/compressible/sonicLiquidFoam/decompressionTank/constant/\
transportProperties $FOAM_RUN/cavity/cavityFirstOrder/constant
```

2.3 Boundary conditions and initial fields

The figure 2.1 shows boundary conditions for the first and second order system of equations.
Initial conditions set to zero for velocity and pressure for the first and second order equations. As shown in the left figure, one of the walls at the top has been considered as a pressure source. This means that a wave is generated at this boundary and propagate into domain. Also, It is assumed pressure gradient at the other walls equals to zero and it means that wave will reflect at these walls.

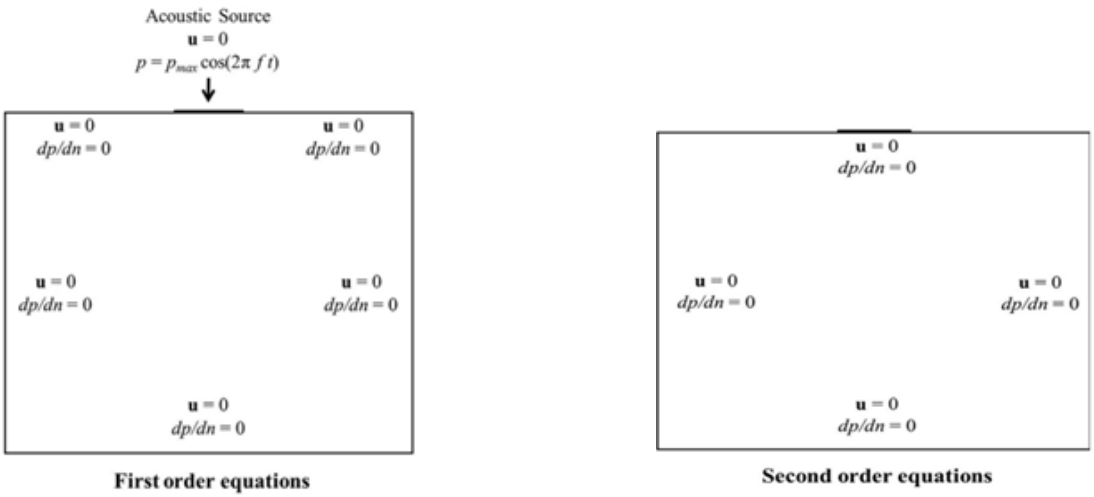


Figure 2.1: Boundary condition for the first and second order equations

Parameter	Value
Sound Speed	1440 m/s
Frequency	10 MHz
Pressure Amplitude	20 kPa
Density	998 kg/m^3
Kinematic viscosity	89 e-08 m^2/s
Time Step–$1^{st} order$	$10^{-9} s$
Simulation time–$1^{st} order$	$\mathrm{O}(10^{-5})\ s$

Table 2.1: Property of fluid and some parameters for numerical modeling

2.4 Property of fluid and simulation parameter

Table 2.1 shows fluid properties and some simulation parameters that are used to simulate first order equations. we should change the value of parameters in `thermodynamicProperties` as follows:

```
rho0            rho0 [ 1 -3 0 0 0 0 0 ] 997;

p0              p0 [ 1 -1 -2 0 0 0 0 ] 100000;

psi             psi [ 0 -2 2 0 0 0 0 ] 4.44e-07;
```

Now, everything is ready to start the simulation. First, go to the case directory and use `blockMesh` to create mesh, Then, write `mySonicLiquidFoam` to start running. It is noteworthy to add `"libOpenFOAM.so"` and `"libgroovyBC.so"` at the end of `controlDict`. As you know, one of them is the library which links the `groovyBc` to the this solver.

2.5 Result (frequency 10MHz)

2.5.1 First order

To validate the code, I simulated a problem with the same specifications in [2]. The value of thermophysical properties and time step are shown in table 2.1. Figure 2.2 shows the result comparison

between [2] and present work. In this figure, the average y-component momentum source has been shown for 4 times (5e-07s, 5e-06s, 8e-06s and 10e-06s) by my code and [2]. Results show good matching between [2] and present work. By comparing the results, I concluded that the average value of momentum source became steady after so I will use the value of all parameters at this time for next results

Figure 2.3 shows the comparison between instantaneous pressure contour in the present work and [2]. Results show good matching between my work and [2]. The instantaneous pressure in the fluid after simulation appears as high and low pressure stripes, as a result of the oscillatory vibration. The propagation of pressure occurs in a semicircular shape with intensity decaying with the distance to the acoustic source. In this case, viscous attenuation has the most important role to attenuate wave intensity so most power of wave is attenuated by the viscosity of fluid before the wave reached to the opposite wall. Also, it is important to note that the number of strips depends on the wave frequency and the speed of wave in the propagation media. In this case, the wavelength is about 0.14 mm and the height of the enclosure is 2 mm therefore we can observe 24 strips by ignoring the viscous attenuation but with considering viscosity we observe less than 24 strips.

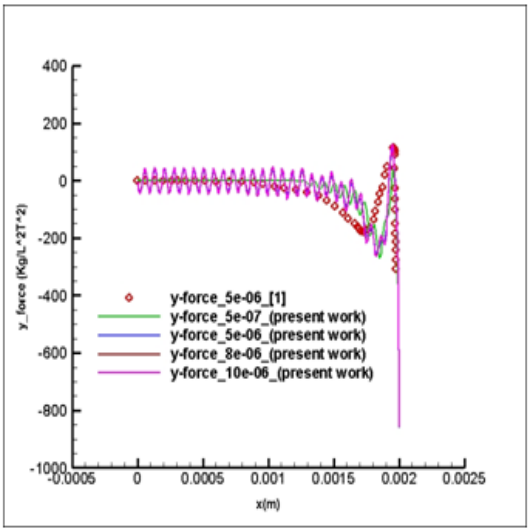
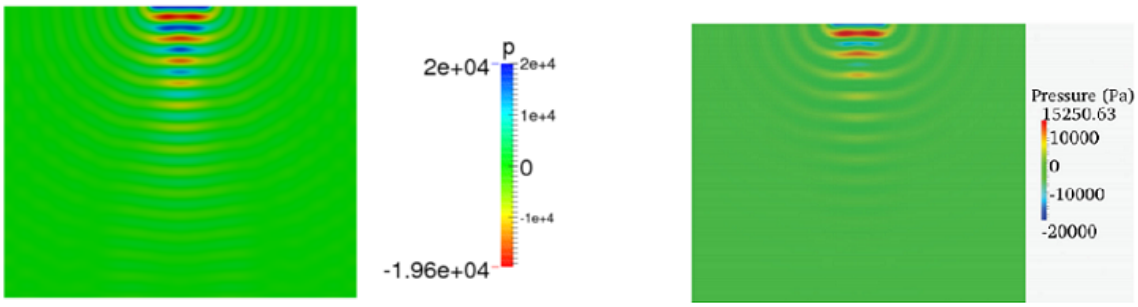


Figure 2.2: average of y-component momentum source at various times



(a) present work (b) [2]

Figure 2.3: Propagation of instantaneous pressure

2.5.2 Second order

Like first order case, it is better to use cavity cases and do all modifications on it. Therefore, copy the cavity tutorial in the run directory and change its name to cavitySecondOrder.

```
mkdir -p $FOAM_RUN/cavity
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN/cavity
cd $FOAM_RUN/cavity
mv cavity cavitySecondOrder
```

Now, change the `blockMeshDict` exactly the same as previous section. Also, we can copy this file from `cavityFirstOrder` and use it for `cavityFirstOrder` case.

```
rm -rf $FOAM_RUN/cavity/cavitySecondOrder/constant/polyMesh
cp -r $FOAM_RUN/cavity/cavityFirstOrder/constant/polyMesh \
$FOAM_RUN/cavity/cavitySecondOrder/constant/polyMesh
```

Now, we remove the `transportProperties` in the `cavitySecondOrder` and copy `transportProperties` from one of the tutorial cases in the buoyantBoussinesqSimpleFoam directory. Also, the `g` file(gravitational acceleration) must be in the `constant` directory. By copying and pasting below lines, All told explanations will be done automatically.

```
rm -rf $FOAM_RUN/cavity/cavitySecondOrder/constant/transportProperties
```

```
cp -p $FOAM_TUTORIALS/heatTransfer/buoyantBoussinesqSimpleFoam/hotRoom/constant\
/g $FOAM_RUN/cavity/cavitySecondOrder/constant
cp -p $FOAM_TUTORIALS/heatTransfer/buoyantBoussinesqSimpleFoam/hotRoom/constant/\
transportProperties $FOAM_RUN/cavity/cavitySecondOrder/constant
```

`buoyantBoussinesqSimpleFoam` solver is used for modeling the buoyancy flow with Boussinesq approximation. Because, we do not want to model this kind of flow, therefore we should assign the gravity (`g`) and `beta` equals to zero.

Figure 2.4 shows the acoustic streaming in the domain. As you see, there are two large vortexes which cause more fluid mixing and decrease the effect of the boundary layer near solid walls. Therefore, we expect that heat transfer enhances due to acoustic streaming.

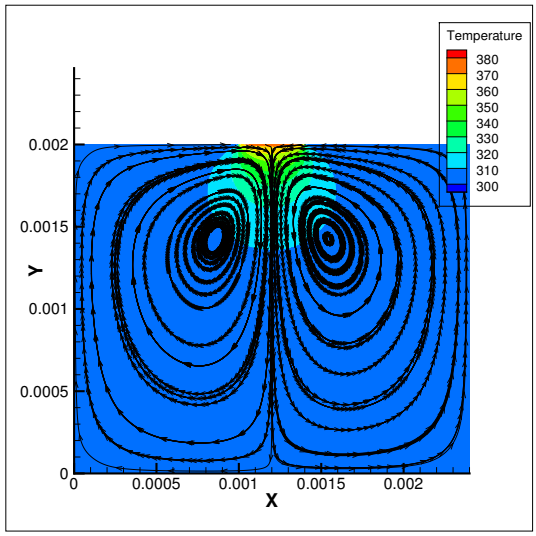


Figure 2.4: Acoustic streaming

Bibliography

[1] W. L. Nyborg Acoustic streaming, Academic Press, San Diego 1998

[2] Catarino, Susana O. and Miranda, Joao M. and Lanceros-Mendez, Senentxu and Minas, Graca, Numerical prediction of acoustic streaming in a microcuvette, The Canadian Journal of Chemical Engineering, Vol, 92, no, 11, 2014