1. Your workings/derivation to show how you started with the governing equation and got to the equation that was implemented in your code

Governing equation:

$$\nabla^2 u + \kappa u = 0 \ \ \& \ \ \kappa = \frac{\rho\omega^2}{\mu}$$

Where $\nabla^2 u = \nabla \cdot \nabla u$

Weighted residual form:

$$\int_\Omega (\nabla \cdot \nabla u + \kappa u)\omega \mathrm{d}\Omega = 0$$

Green-Gauss formula:

$$\int_\Omega (f\nabla \cdot (h\nabla g) + \nabla f \cdot (h\nabla g))\mathrm{d}\Omega = \int_\Gamma f\hbar \frac{\partial g}{\partial n} d\Gamma$$

Substitute $f = \omega$, $g = u$, $h = 1$

Starting equation for FEM

$$\int_\Omega (\nabla\omega \cdot \nabla u - \omega\kappa u)\mathrm{d}\Omega = \int_\Gamma \omega \frac{\partial u}{\partial n} d\Gamma$$

$$\Omega \ = \ \cup_{e=1}^{E} \Omega_e$$

Element stiffness/load, f is the RHS vector

$$K_{mn}u_n = f_m \qquad \begin{cases} K_{mn}u_n = \int_\Omega (\nabla\omega \cdot \nabla u - \omega\kappa u)\mathrm{d}\Omega \\ f_m = \int_\Gamma \omega \frac{\partial u}{\partial n} d\Gamma \end{cases}$$

Let $u = \Psi_n(\xi)u_n$ and $\omega = \Psi_m(\xi)$

Global stiffness matrix K_{mn}

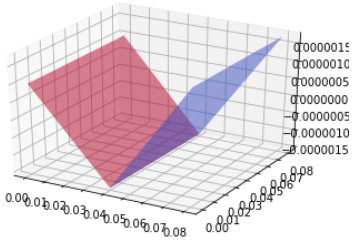
$$K_{mn} = \int_0^1 (\frac{\partial \Psi_m(\xi)}{\partial \xi_i} \frac{\partial \xi_i}{\partial x_k} \frac{\partial \Psi_n(\xi)}{\partial \xi_j} \frac{\partial \xi_j}{\partial x_k} - \Psi_m(\xi)\kappa)J d\xi$$

J is the determinant of $\frac{\partial x_k}{\partial \xi_i}$

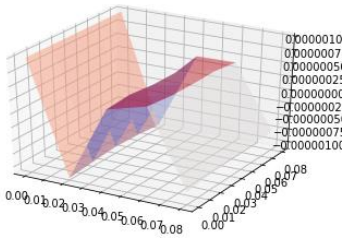
2. Your (well commented) code, including instructions on how to run it if necessary.

3. run your code at 2x2, 4x4, 8x8, 16x16, 32x32 & 64x64 elements per side resolutions.

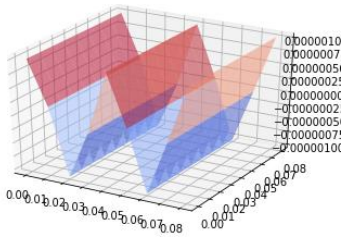
Results of different numbers of elements:



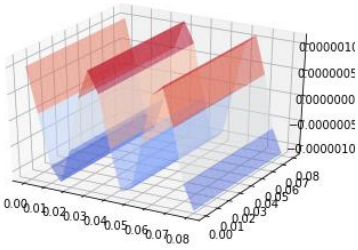
2x2



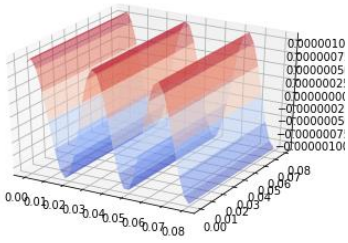
4x4



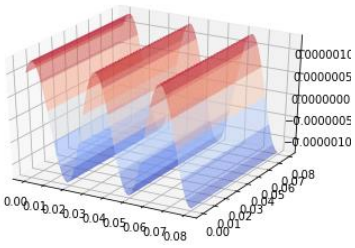
8x8



16x16



32x32



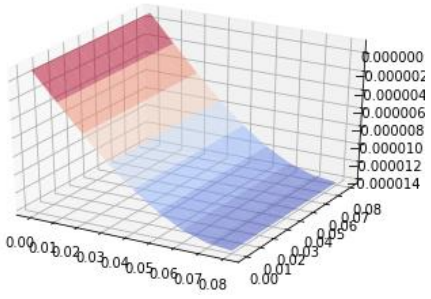
64x64

Regardless run time of the program, 64x64 has the most accurate result.

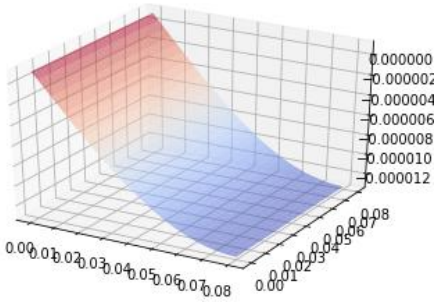
In addition, 16x16 is also an appropriate resolution. Since it has the same tendency with 64x64 and shorter run time.

4. Clinically, the oscillation frequency used is typically 10-1000Hz. Can your implementation get an accurate answer over this range? If yes, justify your conclusion. If not, what you recommend should be done such that it can?

Can get an accurate answer over low frequency. For example, the images below show the results of 10hz at different resolution.



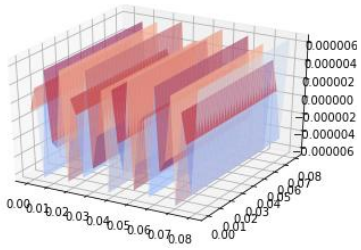
10Hz 8x8



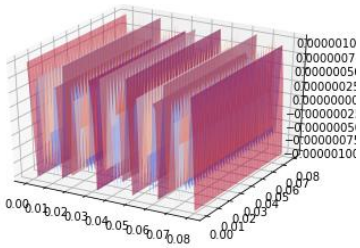
10Hz32x32

However, at high frequency, it is hard to get an accurate answer. For example, there is a big difference between different resolutions (32x32, 64x64 and 80x80) at 1000Hz, which means if the number of elements is not big enough a lot of information is missing.

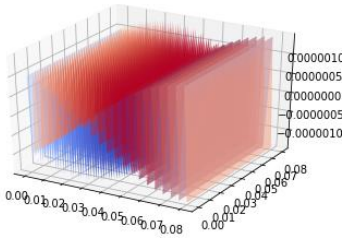
To improve, higher resolution should be used to get a more accurate answer.



1000Hz 32x32



1000Hz 64x64



1000Hz 80x80