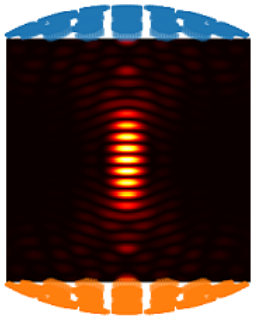
Configurable implementation of a matrix method to simulate pressure in acoustic levitation devices

by
Oskar Paulsson,
University of Gothenburg &
Universidad Nacional Autónoma de México,
guspauosa@student.gu.se
December 9, 2020



Master Thesis in Physics,
Submitted for review to Department of Physics at University of Gothenburg

UNIVERSITY OF
GOTHENBURG

Supervisors: Victor Contreras, Instituto Ciencias Físicas, Universidad Nacional Autónoma de México, victor@icf.unam.mx
Ademir Aleman, Department of Physics, University of Gothenburg, ademir.aleman@physics.gu.se
Examiner: Dag Hanstorp, Department of Physics, University of Gothenburg, dag.hanstorp@physics.gu.se

Abstract

This work presents an implementation of a fast method for simulating acoustic pressure in conventional and novel acoustic levitation devices. The implementation is coded in python, using `numpy` for matrix operations, `matplotlib` for the visualization and `tkinter` is used to develop a graphical user interface. This implementation provides a fast and reliable way to simulate acoustic forces with a user-friendly graphical user interface, broadening the accessibility to this kind of simulation to those in lower levels of education. The software is published as open source to promote independent development. The results of the implementation are validated qualitatively with schlieren imaging and 4 different acoustic levitation devices are evaluated. Three of these devices are derivative of the TinyLev system, a novel device which uses inexpensive components and a chassi which can be 3D printed to create an avenue for democratization of acoustic levivation. The fourth device is a combination of conventional techniques of acoustic levitation, utilizing both a static reflector and a small array of transducer. It is concluded that the matrix method provides a reliable and cost-effective way to simulate the acoustic potential in these kinds of devices.
Keywords: acoustic levitation, the matrix method, TinyLev, schlieren effect, python

Contents

1 Introduction **1**
- 1.1 Brief History and Applications of Acoustic Levitation 2

2 Theory **4**
- 2.1 Nomenclature 4
- 2.2 Wave Mechanics 4
  - 2.2.1 Standing Waves 6
- 2.3 Fluid Dynamics 7
  - 2.3.1 Equation of state 7
  - 2.3.2 Equation of continuity 8
  - 2.3.3 The linear wave equation 8
- 2.4 Wave Solution for the Acoustic Pressure 10
- 2.5 Relative Acoustic Potential 11
- 2.6 Schlieren Effect 11
- 2.7 Numerical Methods for Simulating Acoustic Pressure 14
  - 2.7.1 Matrix Method 14
  - 2.7.2 Lattice Boltzmann Method 16
  - 2.7.3 Far-Field Piston Model 17
  - 2.7.4 Finite Elements Method 18

3 Method **19**
- 3.1 Simulations 19
  - 3.1.1 Matrix Method 19
- 3.2 Acoustic Levitation Device 20
  - 3.2.1 MicroLev 20
  - 3.2.2 MATBig, MATMed and MATSma 22
  - 3.2.3 Design and Circuitry 23
- 3.3 Schlieren Imaging 24
- 3.4 Application Development 24
  - 3.4.1 Rudimentary Implementation 24
  - 3.4.2 Replicating the Geometry of TinyLev 25
  - 3.4.3 Developing the Graphical User Interface 25

4 Results **26**
- 4.1 Generating different geometries 27
- 4.2 Simulations 28
  - 4.2.1 Comparing MATLAB results with the python implementation 28
  - 4.2.2 Simulation results (Python) 31
- 4.3 Schlieren imaging 35
- 4.4 Graphical User Interface 38

5 Discussion **42**
5.1 Matrix Method Implementation in MATLAB and Python 42
5.2 Validation 42
5.3 Error factors in schlieren imaging 46
5.4 Poor alignment in the MicroLev system 50
5.5 Bad geometry 50

6 Conclusion and outlook **52**
6.1 Future Work 53

References **53**

A Code **i**
A.1 Main file i
A.2 GUI i

B The matrix method **xv**
B.1 matrices.py xv
B.2 geometries.py xvii
B.3 computation.py xxi
B.4 MatrixMethod.py xxiii

1 Introduction

Free and easily accessible software (freeware) enables students, researchers and startups to be productive without committing to any financial risk. Open source projects play and important role in the development and maintenance of freeware by enabling any developer to copy and branch off from the original project in order to find novel solutions. Open source projects can also provide great learning opportunities for students by showing concrete real-world examples of how solutions in a particular programming language can look. Python is a suitable entry-level language because it is very high-level but can accomplish complex tasks with very easy-to-read code, relative to for example Java. Python has also become very popular in recent years because of accessibility and high usability when it comes to developing prototype applications.

In recent years, due to the high availability of inexpensive electronics a new device has been invented which can be easily build with the help of a 3-D printer - the TinyLev system is a multi-emitter device which can trap various objects along a single axis in multiple focii [1]. The TinyLev system was developed with the motivation to democratize acoustic levitation. This thesis will follow this motivation to democratize simulation software for systems like the TinyLev.

Following the release of the TinyLev system the Ultraino project was launched by Marzo, Crockett and Drinkwater[2]. It is an open platform for phased-array acoustic levitators with the aim to make available to researchers a tool to simulate phased array systems for ultrasonic applications such as levitation. Phased arrays are collections of multiple elements (transducers) which emit sonic signals at different delay times (phase). The software which they developed is intended for integration with an Arduino board to control the amplitude and phases for an ultrasonic device.

The Ultraino software has a few designs which have been integrated into the program and any other designs has to be imported in the form of 3-D models. To improve accessibility in this regard, this work presents software which has native support to generate some types of geometries which the Ultraino software cannot. These geometries can be flat or concave arrays of transducers with up to 8 "rings" of transducers, with user-specified radius of curvature and flat or concave reflectors which do not emit any signal.

Andrade *et. al* have formulated a matrix method for acoustic levitation simulation [3] which will be used in this thesis to compute acoustic pressure. The method builds on a work done by Ibáñez and colleagues who formulated a monochromatic transfer matrix (MTM) to address non-linear effects involved in the propagation of waves through different media [4]. The MTM method is formulated from the Rayleigh integral which has been used to accurately model the acoustic field between a transducer and a reflector by Kozuka *et. al* [5, 6]. The matrix method has been characterized by comparison with experimental data from a microphone inside a tube, which was placed inside the cavity of a vibrating sonotrode and a reflector [7].

1.1 Brief History and Applications of Acoustic Levitation

Acoustic levitation is the technique of using acoustic radiation forces to counteract gravity to suspend objects in mid-air. Many variations of the technique exists, and it has many modern applications and recent advances have made it possible to manipulate the acoustic field to translate and rotate many different objects in three dimensions [8–12]. One example of stability analysis of an object trapped in an acoustic levitation device was carried out by Baer in 2011 [13].

The history of standing wave acoustic levitation dates back to 1866 with August Kundt [14] who made the observation that fine dust particles accumulate at the pressure nodes of a standing wave which he created in a transparent tube with a resonating rod at one end. With this he demonstrated the acoustic radiation force on small particles due to a standing wave field. Standing wave acoustic levitation is a technique suitable for objects that are much smaller than the acoustic wavelength and can be used to levitate liquid drops and solid objects alike. The first real example of acoustic levitation in a standing wave field came in 1933 by Bücks and Miller [15], they found that alcohol droplets can be suspended in the air if placed at the pressure nodes between a vibrating rod and a reflector. Since its inception, contributions and progress has been plentiful and many variations of acoustic levitation devices have been demonstrated. Some conventional, single-axis methods of acoustic levitation are presented in figure 1.

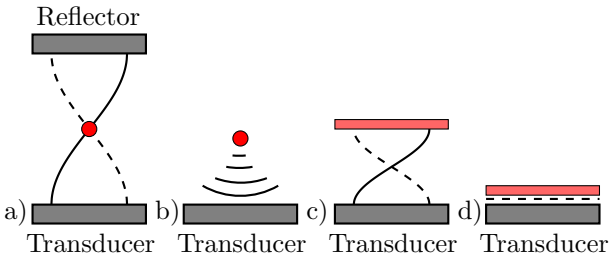


Figure 1: a) Standing wave acoustic levitation, b) Single beam acoustic levitation, c) Far-field acoustic levitation, d) Near-field acoustic levitation. The red dots represents a particle and the red rectangle represents some flat object, real world examples of these levitation techniques can be found in the review by Andrade from 2018 [8].

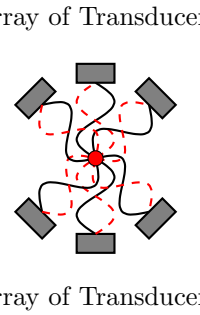


Figure 2: Simplified schematic of a multi-emitter single-axis acoustic levitation device.

Compared to other levitation techniques such as optical, aerodynamic, electronic and magnetic levitation, acoustic levitation is not very restrictive on particle size, state of aggregation or physical properties [16]. A setup of single axis standing wave acoustic levitation device with concave reflector has been of particular interest as they produce a greater axial and radial forces on the object in the trap [17]. This type of setup has been thoroughly studied in terms of particle stability at the nodes [18]. The deformation of droplets in acoustic levitation devices have also been studied [19, 20]. Further, it has been demonstrated in some works that acoustic levitation in general can be used for chemical analysis [21–23]. The TinyLev system specifically has been used for spectral analysis of water droplets containing Cu, Mn, Pb and Ni [24].

Typical acoustic levitation systems found in literature create a trap using a resonant cavity. Such systems require very precisely engineered components and require very high voltages to operate. The TinyLev addresses this problem because it is made of inexpensive materials without compromising on the stability of the levitated object. This is referred to as democratization of technology, where a technology becomes more accessible to more people. The objective of this work is to develop a computerized analytical tool in order to further democratize acoustic levitation.

2 Theory

This section will cover the theoretical principles that govern the interaction between a particle or an object and a gaseous medium (air) which enables us to use an array of sonic emitters to levitate almost any kind of object. The air will be considered to be inviscid, meaning a fluid with no viscosity. Inviscid fluids have fewer constraints on deformations.

This section will also cover methods for computer-simulations of this scenario and how to use an optical effect called the schlieren effect to visualize the acoustic field.

2.1 Nomenclature

Air is considered to be gaseous fluid, therefore kinematic equations of fluids will be set up the to describe the motion of an infinitesimally small volume dV referred to as a fluid element or particle which is large enough to contain millions of air molecules yet small enough that density, pressure, the speed of sound etc. are uniform throughout the element. Its position is given in Cartesian coordinates by the vector $\mathbf{r} = x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}}$ and its velocity by $\mathbf{u} \equiv \dot{\xi} = \nabla\Phi$. Here ξ is some displacement from the equilibrium, Φ is the velocity potential of a spherical wave propagating through the medium and ∇ is the gradient operator. Some quantities that are dealt with will in this section are;

ρ	=	*instantaneous density* at (x, y, z)
ρ_0	=	*equilibrium density* at (x, y, z)
s	=	*condensation* at (x, y, z)
$s\rho_0$	=	$(\rho - \rho_0)$
p	=	*acoustic pressure* at (x, y, z)
c	=	*thermodynamic speed of sound* of the fluid
Φ	=	*velocity potential* of the wave
T_K	=	*temperature* in kelvins (K)
T_C	=	temperature in Celsius
T	=	$T_C + 273.15$
f	=	*frequency* of the sound emitted from some source
ω	=	$2\pi f$: *angular frequency*
λ	=	f/c: *wavelength* of the sound, propagating through a medium with thermodynamic speed of sound c
j	=	$\sqrt{-1}$: *imaginary number*

2.2 Wave Mechanics

Consider a string which is stretched out but at rest. If a portion of the string with length l is displaced from its equilibrium position and released it is known that the disturbance will break up and form two separate disturbances which will propagate in opposite directions, see figure 3.

This type of disturbance is referred to as a transverse traveling wave and is described by the one-dimensional wave equation

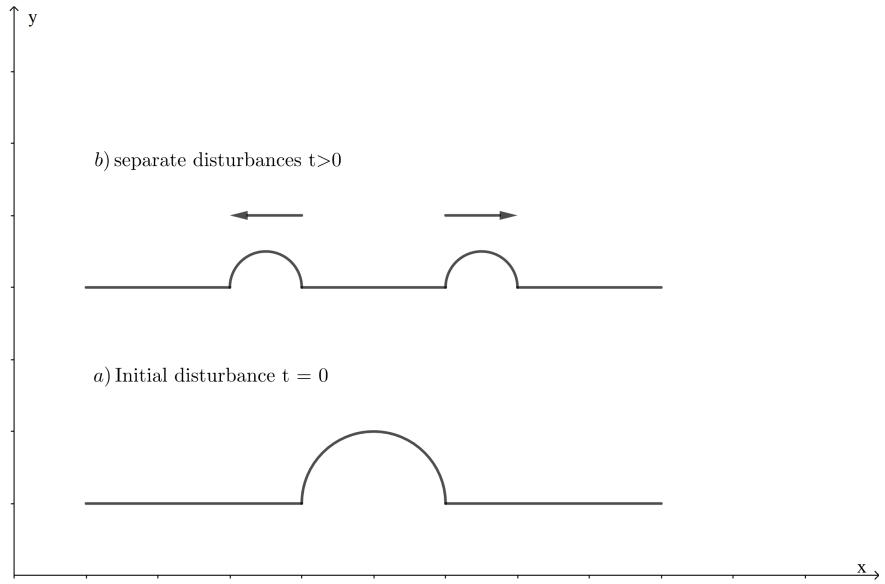


Figure 3: The creation of a transverse wave by displacing a small segment of the string.

$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y}{\partial t^2} \tag{1}$$

with

$$c^2 = T/\rho_L, \tag{2}$$

where T is the tension in the string and ρ_L is the linear density of the string. The general solution for eq. 1 is given by two arbitrary but twice differentiable functions

$$y(x,t) = y_1(ct - x) + y_2(ct + x), \tag{3}$$

where $e^{j\omega(t\pm x/c)}$ is an example of such a function. The complex solution for a monochromatic wave on a string is given by

$$\mathbf{y}(x,t) = \mathbf{A}e^{j(\omega t - kx)} \tag{4}$$

where $A = |\mathbf{A}|$ is the amplitude of the sinusodial wave. Thus, a solution for two waves travelling in opposite directions become;

$$\mathbf{y}(x,t) = \mathbf{A}e^{j(\omega t - kx)} + \mathbf{B}e^{j(\omega t + kx)}. \tag{5}$$

2.2.1 Standing Waves

Consider a string which is fixed in one end and driven in the other. A wave then emanates from the driven part and is reflected at the fixed part. The left end of the string is considered to be the driven part and the relation between the driving force and the amplitude of the wave is given by

$$Fe^{j\omega t} = -T(-jk)\mathbf{A}e^{j\omega t}, \tag{6}$$

where F is the driving scalar force. The boundary condition at the left side of the string becomes

$$Fe^{j\omega t} + T\left(\frac{\partial \mathbf{y}}{\partial x}\right)_{x=0} = 0. \tag{7}$$

The substitution of eq. 5 into the boundary condition 7 gives

$$F + T(-jk\mathbf{A} + jk\mathbf{B}) = 0. \tag{8}$$

The string is rigidly supported at $x = L$ where the displacement is always zero giving

$$\mathbf{A}e^{-jkL} + \mathbf{B}e^{jkL} = 0. \tag{9}$$

Solving eq. 9 for $\mathbf{A}$ and $\mathbf{B}$ leads to the following observation; the wave which emanates from the left side is reflected and because the emanating wave and the reflected wave have the same period but are out of phase and traveling in opposite directions they coincide in such a way that no single wave can be seen propagating in a single direction but the entire string is in constant motion. Such a motion is known as a standing wave and is characterized as having nodes where the string remains at rest, around which the string oscillates. Nodes come at the discrete, even spacing x_n:

$$x_n = L - n\lambda/2 \quad n = 0, 1, 2, \ldots, \leq 2L/\lambda, \tag{10}$$

where x_n is the distance from the driver and L is the length of the string between the driver and the point where the string is fixed [25].

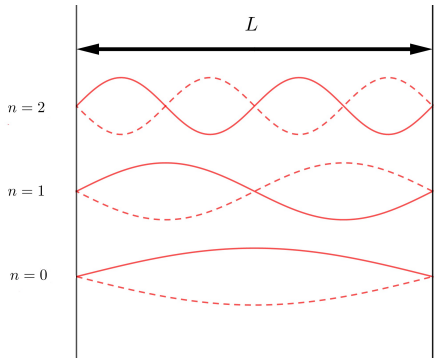


Figure 4: Illustration of standing waves on a string.

2.3 Fluid Dynamics

2.3.1 Equation of state

The equation of state for an ideal gas at constant temperature is given by,

$$P = \frac{n}{V} RT_K = \rho RT_K, \tag{11}$$

where ρ is the density at a location $\mathbf{r}$. P is the total pressure and can be broken into its constituents $P = p + P_0$ where P_0 is the equilibrium pressure and p is the acoustic pressure. Thus

$$p = P - P_0 \tag{12}$$

where $P - P_0$ is found from the Taylor expansion of P:

$$P = P_0 + \left(\frac{\partial P}{\partial \rho}\right)_{\rho_0} (\rho - \rho_0) + \frac{1}{2}\left(\frac{\partial^2 P}{\partial \rho^2}\right)_{\rho_0} (\rho - \rho_0)^2 + ... \tag{13}$$

discarding higher order terms, it is obtained

$$P - P_0 \approx \left(\frac{\partial P}{\partial \rho}\right)_{\rho_0} (\rho - \rho_0) = \rho_0 \left(\frac{\partial P}{\partial \rho}\right)_{\rho_0} \frac{(\rho - \rho_0)}{\rho_0} \tag{14}$$

with $B = \rho_0(\partial P/\partial \rho)_{\rho_0}$ the adiabatic bulk modulus and $s = (\rho - \rho_0)/\rho_0$ the condensation, the acoustic pressure [25] is obtained:

$$p \approx Bs = \left(\frac{\partial P}{\partial \rho}\right)_{\rho_0} (\rho - \rho_0). \tag{15}$$

Thus from eq. 15 the acoustic pressure in terms of the fluid density has been obtained.

2.3.2 Equation of continuity

The equation of continuity is a device to establish a relationship between the velocity of our fluid particles in euclidean space and its compression and expansion (momentary density, ρ). Consider a volume element $dV = dxdydz$ which will be fixed in space and have continuous flow of fluid particles through it. The equation of continuity is given by,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0. \tag{16}$$

By eq. 16 the mass of the fluid is conserved. Furthermore, the gravitational field can be neglected and the fluid is considered to be inviscid so the conservation of momentum as can be written as follows [8, 25],

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right] = -\nabla p. \tag{17}$$

Eq. 17 is also known as Euler's equation.

2.3.3 The linear wave equation

In order to find a solution to the system of equations 15, 16 and 17, some simplifications will be made. First, expand each term to first order approximations by perturbation,

$$p = p_0 + p_1, \tag{18}$$
$$\mathbf{u} = \mathbf{u}_0 + \mathbf{u}_1 = \mathbf{u}_1, \tag{19}$$
$$\rho = \rho_0 + \rho_1. \tag{20}$$

The terms $p_0, \rho_0, \mathbf{u}_0$ refers to an unperturbed system where there is no acoustic wave. In the absence of a wave, the medium is unperturbed and thus $\mathbf{u}_0 = 0$, in the case of small amplitude waves can be simplified further by considering $|p_1| << p_0$ and $|\rho_1| << \rho_0$ so when the expanded terms are inserted into the continuity equation the following is obtained;

$$\frac{\partial(\rho_0 + \rho_1)}{\partial t} + \nabla \cdot ((\rho_0 + \rho_1)\mathbf{u}_1) = \frac{\partial \rho_1}{\partial t} + \nabla \cdot (\rho_0 \mathbf{u}_1) = 0. \tag{21}$$

Since the unperturbed field is constant, $\partial \rho_0 / \partial t = 0$. Because the field is perturbed everywhere at once the velocity $\mathbf{u}$ will change rapidly. Further, the following assumption can be made $|(\mathbf{u} \cdot \nabla)\mathbf{u}| << |\partial \mathbf{u} / \partial t|$ and by taking the divergence of eq. 17, eq. 22 is obtained.

$$\nabla \rho_0 \left(\frac{\partial \mathbf{u}_1}{\partial t} \right) = -\nabla^2 p_1 \tag{22}$$

The term on the left hand side can be retrieved by taking the time derivative of eq 16;

$$\frac{\partial^2 \rho_1}{\partial t^2} + \nabla \rho_0 \left(\frac{\partial \mathbf{u}_1}{\partial t} \right) = 0. \tag{23}$$

At this point, recall that from table 2.1 and equation 20 that $\rho_1 = \rho - \rho_0 = s\rho_0$, thus eq. 23 becomes

$$\rho_0 \frac{\partial^2 s}{\partial t^2} - \nabla^2 p_1 = 0. \tag{24}$$

From 15 $s = p/B$ and since p_0 is the equilibrium pressure its derivative w.r.t time can be neglected. At this point the subscript is dropped so that $p_1 = p$. Then, by introducing isentropic speed of sound in the medium $c_0^2 = B/\rho_0$, eq 25 is obtained.

$$\frac{1}{c_0^2} \frac{\partial^2 p}{\partial t^2} = \nabla^2 p \tag{25}$$

Eq. 25 is the acoustic pressure on the form of the linear and lossless wave equations. Furthermore, since the curl of a gradient must vanish, by eq. 16;

$$\rho_0 \frac{\partial (\nabla \times \mathbf{u})}{\partial t} = -\nabla \times \nabla p = 0 \Rightarrow \nabla \times \mathbf{u} \tag{26}$$

which shows that $\mathbf{u}$ can be expressed in terms of a scalar function as such;

$$\mathbf{u} = \nabla \Phi \tag{27}$$

which was previously identified as the velocity potential. *Physically, this means that the acoustic excitation of an inviscid fluid does not involve any rotational flow.*

Furthermore, the requirement that $\nabla \rho_0 \neq 0$ can be introduced and when eq. 27 is combined with eq. 17 an expression of the acoustic pressure as a solution to the wave equation is obtained

$$\nabla \left(\rho_0 \frac{\partial \Phi}{\partial t} + p \right) = 0 \Rightarrow p = -\rho_0 \frac{\partial \Phi}{\partial t}. \tag{28}$$

Eq. 28 shows that Φ satisfies the wave equation under the approximations were asserted while treating air as an inviscid fluid [8, 25].

2.4 Wave Solution for the Acoustic Pressure

As was seen in eq. 25 the pressure which comes as a result of acoustic excitation can be put in terms of a function. A solution to eq. 25 can therefore be on the complex form, as was shown in section 2.2

$$\mathbf{p} = \mathbf{A}e^{j(\omega t - kx)} + \mathbf{B}e^{j(\omega t - kx)} \tag{29}$$

the particle velocity associated with $\mathbf{p}$ is, by extension of 28 is

$$\vec{\mathbf{u}} = \mathbf{u}\hat{x} = \left[\frac{\mathbf{A}}{\rho_0 c}e^{j(\omega t - kx)} - \frac{\mathbf{B}}{\rho_0 c}e^{j(\omega t + kx)}\right] \tag{30}$$

which is parallel to the direction of propagation. $\mathbf{p}$ can be split into its constituent parts, using the subscripts $+/-$ to denote waves traveling in positive and negative direction along the x-axis respectively,

$$\mathbf{p}_+ = \mathbf{A}e^{j(\omega t - kx)}, \quad \mathbf{p}_- = \mathbf{B}e^{j(\omega t + kx)} \tag{31}$$

$$\mathbf{u}_\pm = \frac{\mathbf{p}_\pm}{\rho_0 c}. \tag{32}$$

When dealing with a plane wave traveling in some arbitrary direction, a solution on the following form is possible

$$\mathbf{p} = \mathbf{A}e^{j(\omega t - k_x x - k_y y - k_z z)} \tag{33}$$

for a plane wave with $k_z = 0$ and with $\vec{k} = k_x\hat{x} + k_y\hat{y}$, $\vec{r} = x\hat{x} + y\hat{y} + z\hat{z}$ gives $\vec{r}\vec{k} = k_x x + k_y y$. There are surfaces of constant phase parallel to the z-axis, their equation is given by

$$y = -(k_x/k_y)x + constant. \tag{34}$$

Because the wavelength λ in the plane can also be broken into λ_x, λ_y with $\lambda_z = 0$, the relationship $\lambda = 2\pi/k$ and $\lambda^2 = \lambda_x^2 + \lambda_y^2$ gives following form for $\vec{k}$

$$\vec{k} = k\cos\varphi\hat{x} + k\sin\varphi\hat{y} \tag{35}$$

where φ is the angle between the x-axis and the normal to the surface line by eq. 34. Substitution with eq. 35 yields the form for the acoustic pressure from plane waves

$$\mathbf{p} = \mathbf{A}e^{j(\omega t - kx\cos\varphi - ky\sin\varphi)}. \tag{36}$$

The form of the acoustic pressure from spherical waves is [25]

$$\mathbf{p} = \frac{\mathbf{A}}{r}e^{j(\omega t - kr)}. \tag{37}$$

2.5 Relative Acoustic Potential

In his paper from 1977 [26], Beyer recounts the history of the study of waves and motivates the existence of acoustic radiation pressure by the similarity between sound and light. His opening argument is that if there is a radiation pressure resulting from the wave-nature of photons, there must be an acoustic radiation pressure as a result of a sound wave. Beyer concludes that the Rayleigh radiation pressure is entirely non-linear. To obtain a solution which takes non-linearity into account an expansion of equations 1.8-1.10 is needed:

$$p = p_0 + p_1 + p_2, \tag{38}$$

$$\mathbf{u} = \mathbf{u}_1 + \mathbf{u}_2, \tag{39}$$

$$\rho = \rho_0 + \rho_1 + \rho_2. \tag{40}$$

From here, Andrade [8] derives the force on an object in an acoustic pressure field to be

$$\mathbf{F}_{rad} = -\int_{S_0} \langle p_2 \rangle \mathbf{n} dS - \int_{S_0} \rho_0 \langle (\mathbf{n} \cdot \mathbf{u}_1)\mathbf{u}_1 \rangle dS, \tag{41}$$

with $\mathbf{n}$ being the normal vector to the surface element dS of a particle in the acoustic field. Equation 41 can be expressed in terms of a potential according to Gor'kov theory

$$\mathbf{F}_{rad} = -\nabla U, \tag{42}$$

where

$$U = 2\pi R^3 \left[\frac{f_1}{3\rho_0 c_0^2} \langle (p_1^{in})^2 \rangle - \frac{f_2 \rho_0}{2} \langle \mathbf{u}_1^{in} \cdot \mathbf{u}_1^{in} \rangle \right] \tag{43}$$

which is the same term that is being used in the matrix method for acoustic levitation [3]. This term is the primary objective of the simulations presented in this report. From eq. 43 the relative acoustic potential may be obtained as follows,

$$\tilde{U} = \frac{U}{2\pi R^3}. \tag{44}$$

2.6 Schlieren Effect

The effect of schliere (german for streaks or striae) is a phenomena whereby light is deflected in the presence of a non-zero gradient of the refractive index n over the cross-sectional area through which the light is passing. Air is considered to have a refractive index of 1 so any medium is either more refractive or less

refractive than air, having a refractive index greater or lesser than 1 respectively. Phenomena of this kind has been studied since the 17th century since its formal origination by Robert Hooke. Schlieren effects can be an aid in the study of the flow and heat transfer in gases by visualizing the changes in refractive index. A fundamental relationship in the schlieren effect is the Gladstone-Dale relation between the density of a gas and the refractive index [27]:

$$n - 1 = k\rho. \tag{45}$$

The Gladstone-Dale coefficient k is mostly constant for visible light, with an approximate value of $2.3 \cdot 10^{-4}$ m^3/kg for air. Snell's law determines how many degrees a beam of light is deflected from it's original path when crossing the boundary between two medium with different refractive index:

$$\frac{n_2}{n_1} = \frac{\sin\theta_2}{\sin\theta_1}. \tag{46}$$

Snell's law is appropriate for situations where light travels through two distinct media with different density but in the case of an acoustic levitation device the medium is the same and the density in the cavity changes gradually in some regions. Consider instead the following relation,

$$\delta = kL\frac{d\rho}{dx}. \tag{47}$$

δ is a small deviation from the original pathway of a light-beam, after it has traversed a region where the gradient $d\rho/dx$ is non-zero, k is the Gladstone-Dale coefficient, L is the span of the of the disturbance on the optical axis and the x-axis is perpendicular to the optical path. The sensitivity of the schlieren imaging technique is known in terms of how much of the unrefracted light is blocked. If more light is blocked the background becomes darker and the schliere are easier to discern. An approximate relation for the sensitivity is given by:

$$\frac{\Delta A}{A} = \frac{\Delta I}{I} = \frac{3\delta L_f}{h}. \tag{48}$$

In equation 48 L_f is the focal length of the mirror, δ is the deflection angle, A and ΔA are the areas of the image which is unobstructed and the area of the image which is being blocked respectively.

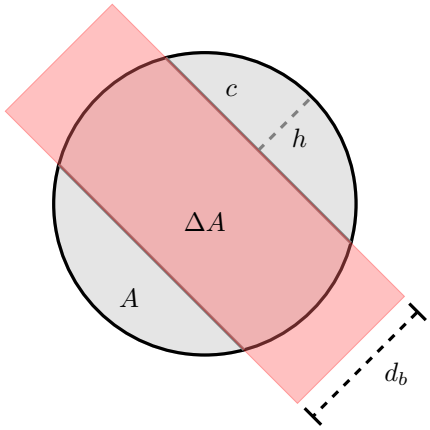


Figure 5: Geometry of the image that is projected from the spherical mirror.

Figure 5 shows the geometry of the image of the mirror where light is partially obstructed, the red area represents the area of obstructed light. Figure 6 illustrates the full geometry of the schlieren setup. A and ΔA are approximately $A = (2/3)ch$ and $\Delta A = cd$.

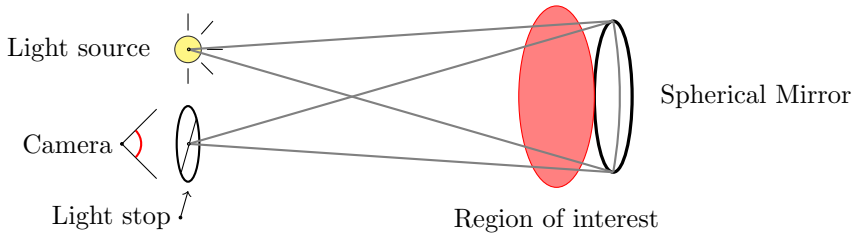


Figure 6: Schematic over the schlieren setup used in this work.

2.7 Numerical Methods for Simulating Acoustic Pressure

This section investigates and reviews different techniques that can be used to compute pressure in an acoustic levitation device.

2.7.1 Matrix Method

The matrix method for acoustic levitation simulation combines Gor'kov theory [28] with a method of computing *transfer matrices* to compute the acoustic potential field by simulating propagation of a sound wave as it is emitted by a transducer, excites a medium, reflects once from a reflector then reflects again on the transducer and so on [3].

The pressure matrix $\mathbf{P} = [p_1, p_2, ..., p_M]^T$ according to the matrix method which is schematically displayed in figure 7.

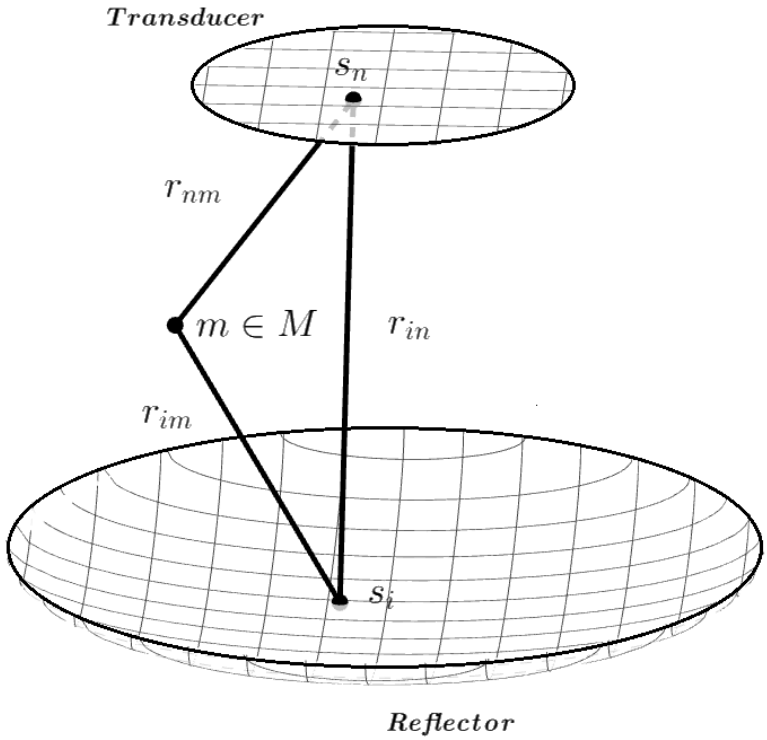


Figure 7: Schematic of the geometry of the matrix method.

The transfer-matrices are $\mathbf{T}^{(TM)}$, $\mathbf{T}^{(RM)}$, $\mathbf{T}^{(TR)}$ and $\mathbf{T}^{(RT)}$ are given by

$$T_{mn}^{(TM)} = s_n \frac{\exp(-jkr_{nm})}{r_{nm}}, \tag{49}$$

$$T_{in}^{(TR)} = s_n \frac{\exp(-jkr_{in})}{r_{in}}, \tag{50}$$

$$T_{ni}^{(RT)} = s_i \frac{\exp(-jkr_{in})}{r_{in}}, \tag{51}$$

$$T_{mi}^{(RT)} = s_i \frac{\exp(-jkr_{im})}{r_{im}}. \tag{52}$$

Where n denotes the discretized elements of the transducer with area s_n, i denotes the elements of the reflector of area s_i, m denotes the index of each point in the plane where the acoustic potential will be calculated (measurement plane). r_{nm}, r_{im} and r_{in} are elements of the matrices which contain the distances between points in the measurement plane and transducer, measurement plane and reflector and transducer and reflector respectively. The expression to calculate $\mathbf{P}$ is given by;

$$\begin{aligned} \mathbf{P} = A \cdot \mathbf{T}^{(TM)}\mathbf{U} + A \cdot C \cdot \mathbf{T}^{(RM)}\mathbf{T}^{(TR)}\mathbf{U} \\ + A \cdot C^2 \cdot \mathbf{T}^{(TM)}\mathbf{T}^{(RT)}\mathbf{T}^{(TR)}\mathbf{U} \\ + A \cdot C^3 \cdot \mathbf{T}^{(RM)}\mathbf{T}^{(TR)}\mathbf{T}^{(RT)}\mathbf{T}^{(TR)}\mathbf{U} \\ + A \cdot C^4 \cdot \mathbf{T}^{(TM)}\mathbf{T}^{(RT)}\mathbf{T}^{(TR)}\mathbf{T}^{(RT)}\mathbf{T}^{(TR)}\mathbf{U} + ... \end{aligned} \tag{53}$$

with

$$\begin{aligned} A &= \frac{\omega \rho c}{\lambda}, \\ C &= \frac{j}{\lambda}, \\ U_i &= \exp\left(-j\omega t_i\right) \end{aligned}$$

A term out of the expression for $\mathbf{P}$ could look like

$$A \cdot \mathbf{T}^{(TM)}\mathbf{U} = \frac{\omega \rho c}{\lambda} \begin{bmatrix} T_{11}^{(TM)} & T_{12}^{(TM)} & \dots & T_{1N}^{(TM)} \\ T_{21}^{(TM)} & T_{22}^{(TM)} & \dots & T_{2N}^{(TM)} \\ \vdots & \vdots & \ddots & \vdots \\ T_{M1}^{(TM)} & T_{M2}^{(TM)} & \dots & T_{MN}^{(TM)} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_N \end{bmatrix}. \tag{54}$$

When the pressure has been calculated, the real part $p = \mathrm{Re}\left(\mathbf{P}\right)$ is used and Gor'kov theory is applied to compute the acoustic potential;

$$V = 2\pi R^3 \left(\frac{\overline{p^2}}{3\rho c^2} - \frac{\rho \overline{\dot{u}^2}}{2} \right) \tag{55}$$

with $\overline{p} = p/M$ with M being the number of points in the measurement plane so $\overline{p}$ is the mean amplitude and $\overline{p^2}$, $\overline{\dot{u}^2}$ are the mean squared amplitudes. The velocity fields are obtained by

$$\phi = -\frac{p}{j\omega\rho} \tag{56}$$

$$\dot{u} = \nabla\phi. \tag{57}$$

The relative acoustic potential becomes

$$\tilde{V} = \frac{V}{2\pi R^3} \cdot f \tag{58}$$

This expression enables us to calculate the acoustic potential without the dependence of an object inside the cavity. Note that eq. 55 is the same as eq. 43 with non-linearity terms, i.e. $f_1, f_2 = 1$. Therefore can be concluded that the method presented in this subsection does not account for non-linear effects. We can also conclude that p is the same as p_1^{in} which is the incident pressure on an object in the acoustic field. Finally, the quantity which will be examined in this work is referred to as the acoustic radiation pressure and was derived by Wang [29],

$$p_{rad} = \frac{p^2}{4\rho_0 c^2}. \tag{59}$$

2.7.2 Lattice Boltzmann Method

The lattice Boltzmann method (LBM) has previously been applied to fluid streaming [30], forces on a cylinder in a sound field [31] and the dynamics of an acoustically levitated particle [32]. In a lattice Boltzmann model, particle motions are simulated by letting them move on a grid. Different grid-models exists throughout the literature on LBM.

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta x, t + \Delta t) = f_i(\mathbf{x}, t) + \Omega_i(f(\mathbf{x}, t)) \tag{60}$$

Equation 60 is the lattice Boltzmann Equation with the lattice gas automaton (LGA) Ω_i, which is the collision operator and represents the rate of change in the particle velocity, f_i from collisions [33]. The LGA is a type of cellular

automaton which can be used to derive the macroscopic Navier-Stokes equations [34]. Readers who are unfamiliar with cellular automata are referred to the following videos for a short introduction [35–37]. In short, cellular automata are generative algorithms which produce binary sequences according to a certain rule-set with respect to a neighbourhood of sites, such as the von Neumann neighbourhood [38]. This type of neighbourhood is also present in the LBM and the direction to adjacent cells is denoted in eq. 60 by $\mathbf{e}_i$.

The collision term in eq. 60 can be approximated as

$$\Omega_i \approx -\frac{1}{\tau}\left[f_i(\mathbf{x},t) - f_i^{(eq)}(\mathbf{x},t)\right], \tag{61}$$

where $f_i^{(eq)}(\mathbf{x},t)$ is the local equilibrium distribution function and τ is the relaxation time for each site on the grid. The local equilibrium distribution function is given by:

$$f_i^{(eq)}(\mathbf{x},t) = \omega_i \rho \left[1 + 3\mathbf{e}_i \cdot \mathbf{u}\Delta x + \frac{9}{2}\left(\mathbf{e}_i \cdot \mathbf{u}\Delta x\right)^2 - \frac{3}{2}u^2\right], \tag{62}$$

where ω_i is the angular frequency at the site, ρ and $\mathbf{u}$ are the density and velocity defined by

$$\rho(\mathbf{x},t) = \sum_{i=0}^{8} f_i(\mathbf{x},t), \quad \mathbf{u}(\mathbf{x},t) = \frac{1}{\rho}\sum_{i=0}^{8} f_i(\mathbf{x},t)\mathbf{e}_i\Delta x. \tag{63}$$

In the case of an acoustic levitator, the term $3\omega_i P c_{iy}$ is added to eq. 60 at the sites of the surface of the transducer. In a report from 2008, Barrios and Rechtmann use LBM to simulate the dynamics of a particle in two different acoustic simple levitation devices, one having a flat reflector and the other having a concave reflector [32]. A principal finding of theirs was that a concave reflector is more efficient than a flat reflector, with the price for this efficiency being that the particle exhibits more complex dynamics. Furthermore, who conclude that the LBM can be extended to work with more complicated geometries.

2.7.3 Far-Field Piston Model

A far field piston model is relatively simple and does not account for non-linear effects and reflections but is an appropriate model for real-time calculations. The complex acoustic pressure is given by

$$P(\mathbf{r}) = P_0 A \frac{D_f(\theta)}{d} e^{j(\varphi + kd)}, \tag{64}$$

where $\mathbf{r}$ is the distance from the piston-source which is emitting a soundwave with frequency f. P_0 is a constant which defines the transducer amplitude

power, A is the peak-to-peak amplitude of the excitation signal and d is the propagation distance in free space where the term $1/d$ accounts for divergence.

$$D_f = \frac{2J_1(ka\sin\theta)}{ka\sin\theta} \tag{65}$$

is the far-field directivity function of a cirular piston source. In eq. 65 a is the radius of the transducer, k is the wave number and J_1 is the first-order Bessel function of the first kind. For configurations with multiple sources, the complex pressure is simply calculated by summing one term per acoustic source.

The Gor'kov potential is used to compute the acoustic force where the potential for the Gor'kov eq. $F = -\nabla U$ is given by

$$U = 2K_1(|p|^2) - 2K_2(|p_x|^2 + |p_y|^2 + |p_z|^2) \tag{66}$$

with

$$K_1 = \frac{1}{4}V\left(\frac{1}{c_0^2\rho_0} - \frac{1}{c_s^2\rho_s}\right)$$
$$K_2 = \frac{3}{4}V\left(\frac{\rho_0 - \rho_s}{\omega^2\rho_0(\rho_0 + 2\rho_s)}\right).$$

Here, V is the volume of a spherical object which is being levitated, ω is the particle of the emitted waves, ρ is the density and c_0 is the speed of sound in the medium while c_s is the speed of sound through the material of the particle. An example of the values for these parameters can be found in the Ultraino-report [2]. p is the complex pressure and p_x, p_y, p_z are the spatial derivatives of p in the respective directions.

2.7.4 Finite Elements Method

Finite elements method (FEM) is a numerical method for solving systems of non-linear partial differential equations that cannot be solved analytically. The computations are usually carried out by some commercially availiable software such as COMSOL or ANSYS. Users have to determine what functions they want the program to solve and which boundary conditions to use so the application will vary on a case by case basis. The main idea of FEM is that to solve the system, constituent parts of the system (e.g. surfaces, bodies and interfacing medium) are discretized into very small, finite parts. FEM has been used to solve the acoustic radiation force on an arbitrary particle and planar reflector [39, 40] and the potential inside a Langevin-type acoustic levitation device [41].

3 Method

3.1 Simulations

3.1.1 Matrix Method

In this work the matrix method has been chosen to simulate the acoustic radiation pressure. It is a convenient choice because it does not simulate a specific particle in the acoustic trap. The implementation was coded first in MATLAB and then in Python. The initial implementation in MATLAB was validated by comparing the results with experimental results found in the literature. The results of the Python implementation was then compared to those of the MATLAB implementation.

The results of the simulations were validated by using schlieren imaging to visualize the acoustic potential and then counting the number of nodes present in the schlieren images and in the simulation of the acoustic radiation pressure to see if they match.

In this work the matrix method presented in section 2.7 was modified to better approximate the geometry of the TinyLev system. The arrays of transducers are considered to be completely non-reflective as their shape is very un-even, albeit symmetrical so the wave will scatter and its energy will dissipate rather than reflect. Therefore in the case of two arrays, only the first terms of eq. 53 will be considered

$$\mathbf{P}^T = A \cdot \mathbf{T}^{(TM)}\mathbf{U}, \tag{67}$$

and

$$\mathbf{P}^B = A \cdot \mathbf{T}^{(RM)}\mathbf{U}. \tag{68}$$

The superscripts T and B stands for top and bottom, respectively. Thus $\mathbf{P}$ becomes

$$\mathbf{P} = \mathbf{P}^T + \mathbf{P}^B. \tag{69}$$

In the case where the top or bottom component is a reflector and the other component is an array or a transducer, the expressions for $\mathbf{P}^T$ and $\mathbf{P}^B$ are also different. An example, with T an array and B a reflector $\mathbf{P}^T$ and $\mathbf{P}^B$ become

$$\mathbf{P}^T = A \cdot \mathbf{T}^{(TM)} \cdot U + A \cdot C \cdot \mathbf{T}^{(RM)}\mathbf{T}^{(TR)}\mathbf{U} \tag{70}$$

$$\mathbf{P}^B = \emptyset. \tag{71}$$

The wave from the array reflects once and then scatters on the array and the reflector does not actively contribute to the total pressure in the trap.

3.2 Acoustic Levitation Device

All of the acoustic levitation devices used in this work operate at and excitation frequency of 40KHz. The acoustic levitation devices used in these experiments were assembled according to publicly available instructions [42]. The electrical components were inexpensive and are commercially available.

3.2.1 MicroLev

The MicroLev is very small acoustic levitation device which features an array of 17 transducer and a concave reflector. The structural parts of the MicroLev which hold the transducers in place were machined by the on-site CNC-mill at the ICF-UNAM. Plans for the structural parts were provided by Alan Reyes, fellow student at UAEM. The reflector was constructed at Gothenburg University.

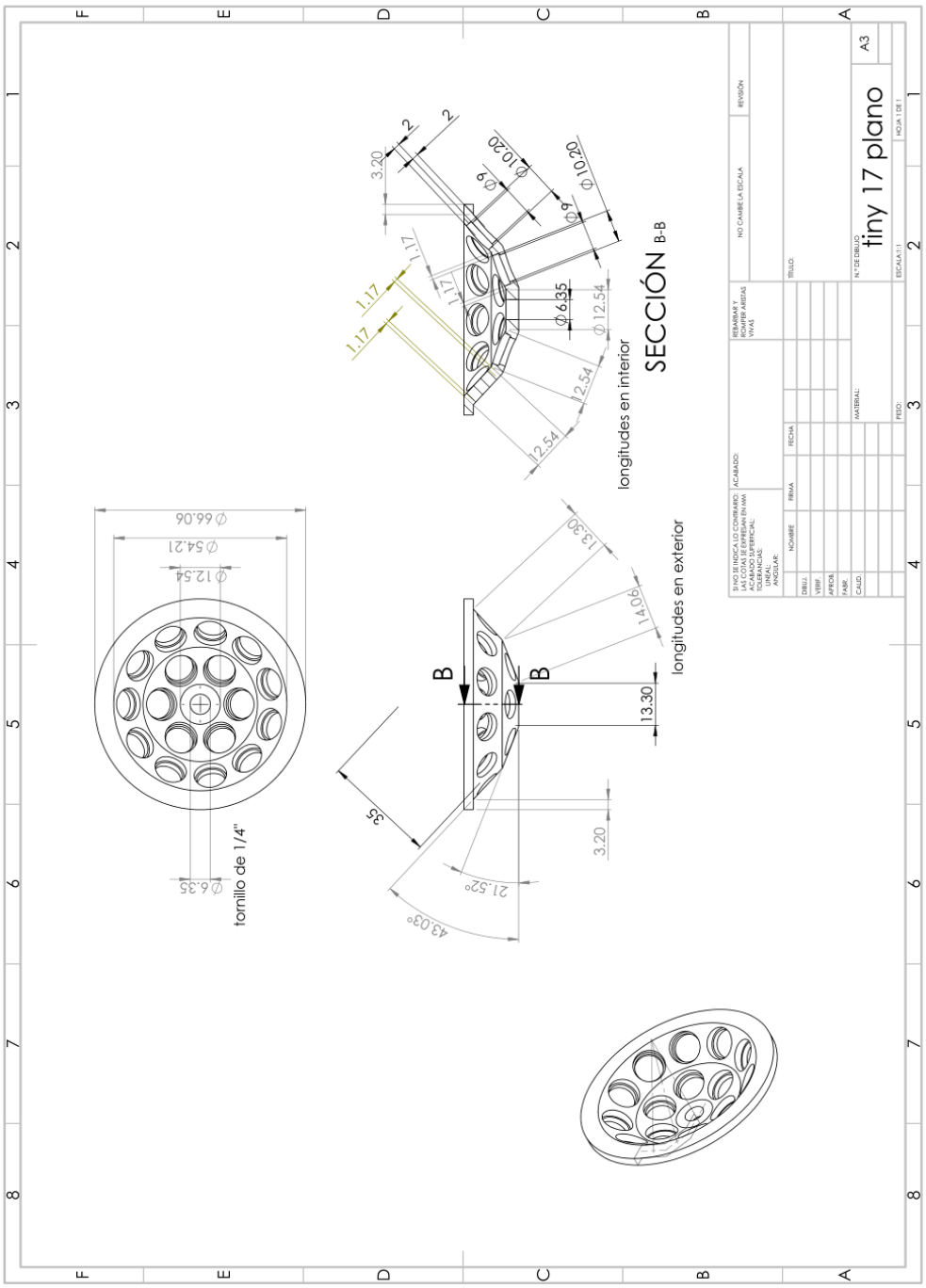


Figure 8: Drawing of the MicroLev array, credit to Alan Reyes of UAEM.

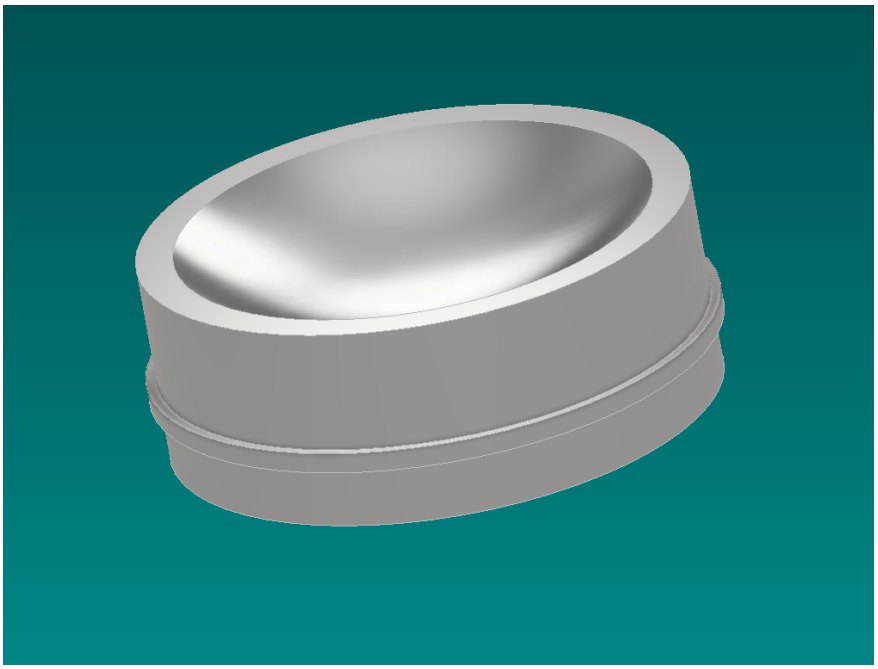


Figure 9: CAD-drawing of the MicroLev reflector, credit to Jan-Åke Wiman of University of Gothenburg.

3.2.2 MATBig, MATMed and MATSma

The construction of these devices are completely credited to Johan Jellstam [43] who worked on a similar project in the months prior. The devices were nicknamed in order of size MATBig, MATMed and MATSma and they were made with the 3D printer at Chalmers University: A sketch of these devices are shown in figure 10 and their features such as cavity length, radius of curvature and number of transducers are given in table 1.

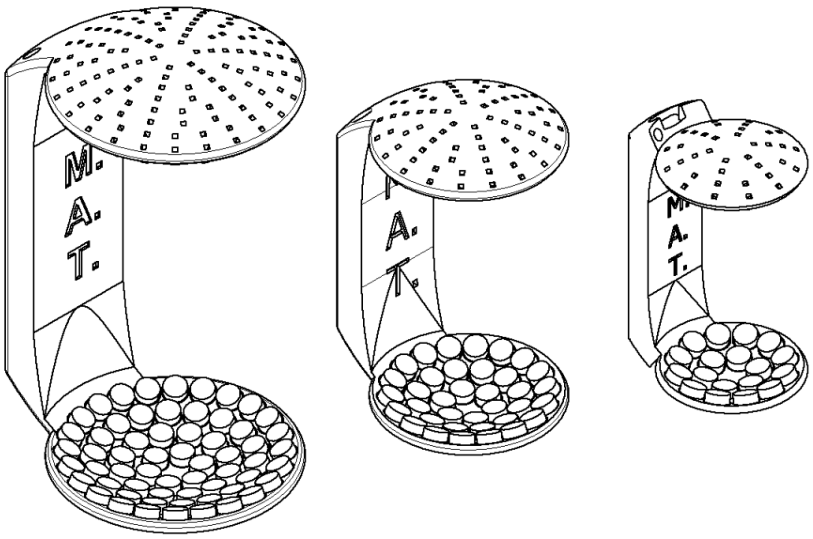


Figure 10: Schematic of the MATs, made by Johan Jellstam.

	Cavity Length (mm)	Radius of curvature (mm)	Number of transducers
MATBig	205	102.5	120
MATMed	128	64	70
MATSma	117	58.5	36
MicroLev	66	35 / 33	17

Table 1: Characteristic parameters of each of the acoustic levitation devices used in this work. For the MicroLev system 35mm radius of curvature refers to the array of transducers while 33mm refers to the reflector.

3.2.3 Design and Circuitry

The circuitry of the acoustic levitator has three necessary components: an array of transducers, a power supply to drive the transducers and a controller for the power supply. In this work an Arduino nano was used as a controller, running a script which can be found here [42].

The design of structural part of the MicroLev which holds the transducers was obtained by numerous trials made by prof. Victor Contreras of The National University of Mexico (UNAM). There are fewer transducers (17) than most other designs which has been seen in demonstrations of acoustic levitation and features a reflector on the bottom. This setup has less instability partially because there are fewer transducers which each provide some degree of instability because they are imperfect, in other words they are not ideal circuits and they are bound to have a phase difference between them.

3.3 Schlieren Imaging

The schlieren setup for imaging consisted of the following items: 1) DSLR-camera, 2) spherical mirror with a focal length of $L_f = 1$ (m), 3) a $d_b = 600$ (μm) diameter wire used for blocking the incoming light from the mirror, 4) a fiber-optic cable with a core diameter of $d_l = 400$ (μm). In order to capture schlieren images of the acoustic potential, the acoustic levitation device was placed in front of the mirror and the light blocker placed in parallel with the focal plane of the mirror, at a distance $l = 2L_f$ from the mirror. The light blocker was attached to a translation-stage which enable micrometer adjustment of its position in the parallel and perpendicular directions to the focal plane.

The setup was configured by first adjusting the mirror so that the image of the mirror in the area of the light blocker. The light blocker is then adjusted so that the image falls directly on it thereby blocking the maximal amount of light. Figure 11 shows the principal geometry of the setup.

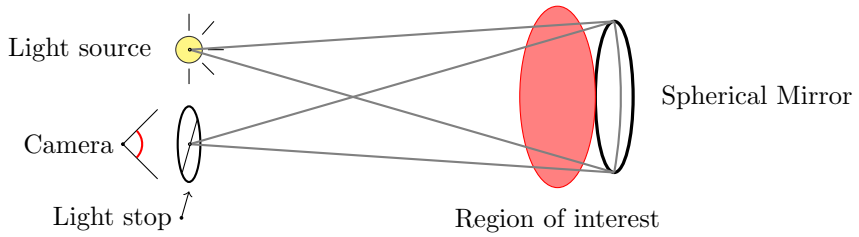


Figure 11: Schematic over the schlieren setup used in this work.

3.4 Application Development

The development cycle of the simulation script and the graphical user interface to control the simulation script was broken into several parts, as explained below.

3.4.1 Rudimentary Implementation

Many researchers have previously examined the acoustic potential inside acoustic levitation devices, most commonly using Langevin-type devices. Modelling of the acoustic potential has been performed both numerically and experimentally by other researchers. After writing a rudimentary script of the matrix method for simulating acoustic potential the results of the script were compared to previous work by other researchers to verify that the implementation was correct.

The geometries that were simulated at this stage were of a flat transducer with a flat reflector and a flat transducer with a concave reflector. This implementation could handle variations on the radius of the transducer and reflector separately and the radius of curvature of the reflector alone.

3.4.2 Replicating the Geometry of TinyLev

A script was coded in MATLAB to generate points which were organized to replicate the surface of each transducer in a TinyLev-system. This script was then evaluated and modified according to compatibility with general specifications. The goal was in principle to produce a script which could take a number of highly relevant parameters, specified by the user which determine the size and shape of the acoustic levitation device which they wish to simulate.

The most relevant parameters were deemed to be radius of curvature on the assembly of transducers, radius of the structural aperture which houses the array, distance from the the zero position, the number of transducers in the arrangement by first determining how many rings the array should have and then how many transducers should be in each ring.

3.4.3 Developing the Graphical User Interface

Beyond the parameters mentioned above the script that runs the simulation should be able to handle the output-frequency and phase of the transducers. The design of the graphical user interface should be intuitive to use and of a "what you see is what you get"-nature.

4 Results

In this section results are presented which were obtained by running the matrix method, the schlieren images that were taken to validate the results of the simulations and the graphical user interface which wraps the simulation algorithm.

4.1 Generating different geometries

The algorithm for evaluating the matrix method includes a method for generating specific geometries of different varieties. Some of those geometries are presented here.

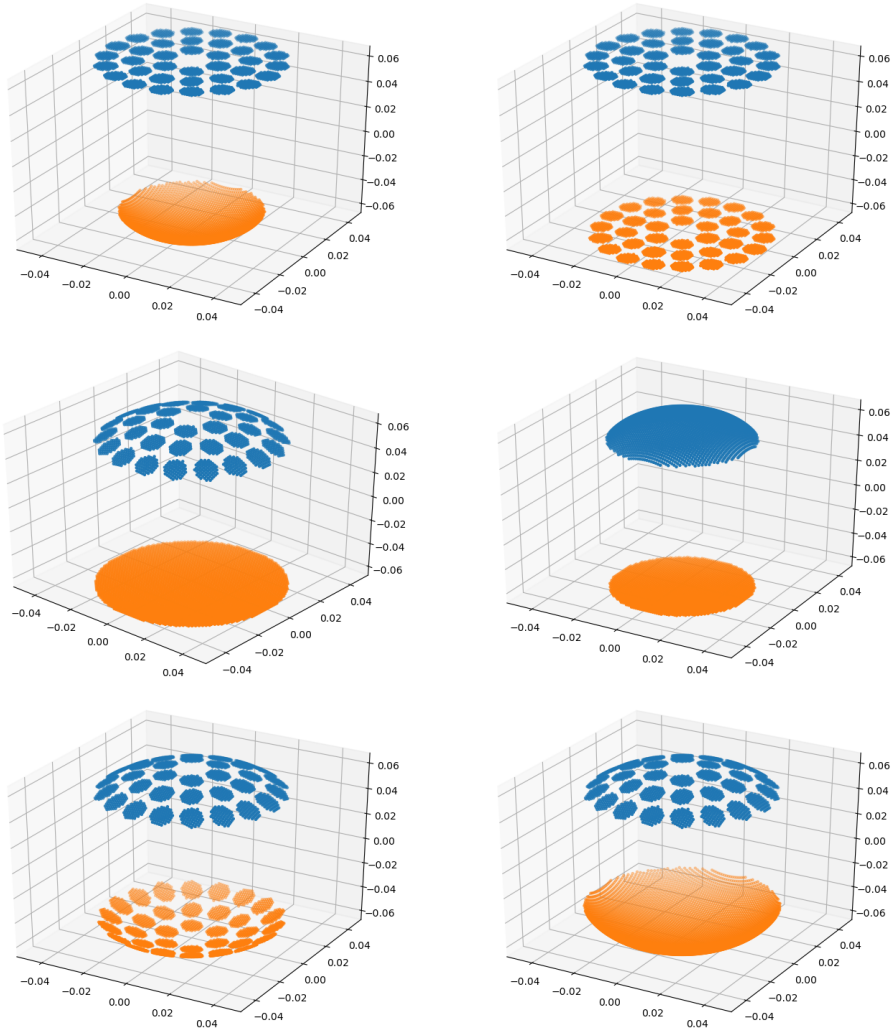


Figure 12: A variety of sample geometries that the program can generate.

4.2 Simulations

This section presents results which were obtained by running the simulation for the parameters defined by the MATBig, MATMed, MATSma and the MicroLev systems as well as the schlieren images that were taken to validate the results of the simulations. The results of running the matrix method in python were exported to a text file and subsequently imported in MATLAB to produce the plots in figures 13 to 18. Section 4.2.2 shows the results that are generated by the GUI, the graphics in figures to are generated using `matplotlib` with default settings, hence the difference in background and aspect ratio from earlier figures.

4.2.1 Comparing MATLAB results with the python implementation

This section presents a comparison between the implementation of the matrix method using MATLAB and Python respectively to show that the Python libraries are able to generate results that are qualitatively identical to MATLAB.

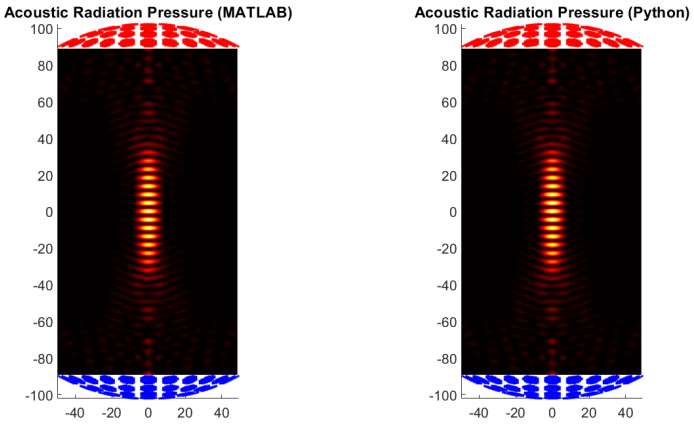


Figure 13: Side by side comparison of the result of simulating the "MATBig" with the matrix method implemented in MATLAB and python (left and right respectively)

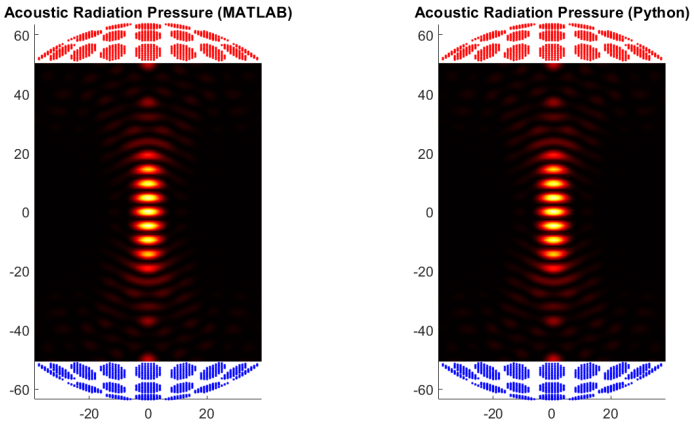


Figure 14: Side by side comparison of the result of simulating the "MATMed" with the matrix method implemented in MATLAB and python (left and right respectively)

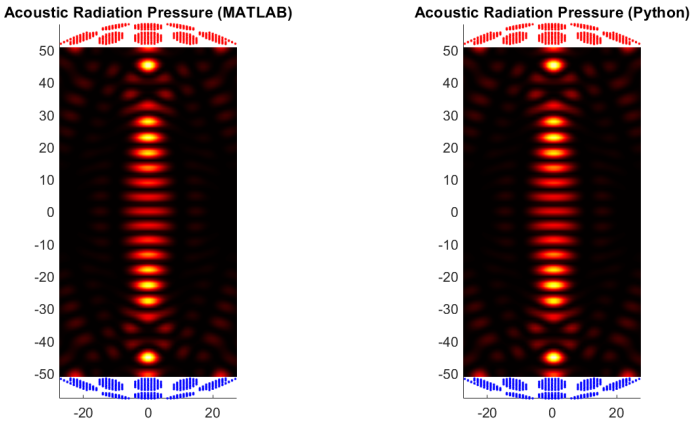


Figure 15: Side by side comparison of the result of simulating the "MATSma" with the matrix method implemented in MATLAB and python (left and right respectively)

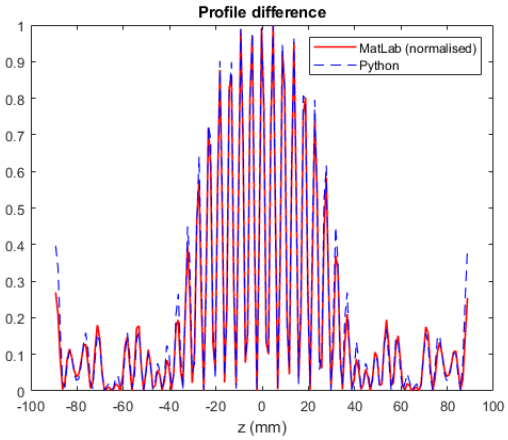


Figure 16: The normalized profile curves of the plots show in figure 13 with MATLAB in red and python in blue.

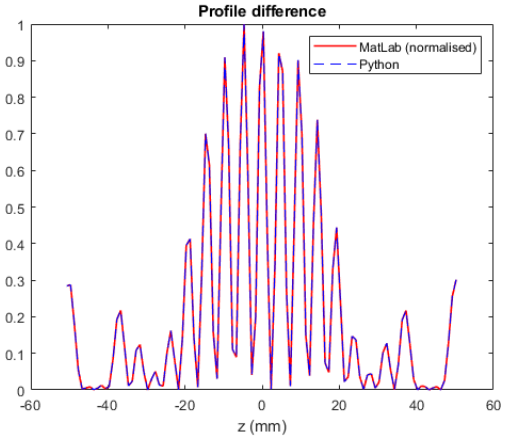


Figure 17: The normalized profile curves of the plots show in figure 14 with MATLAB in red and python in blue.

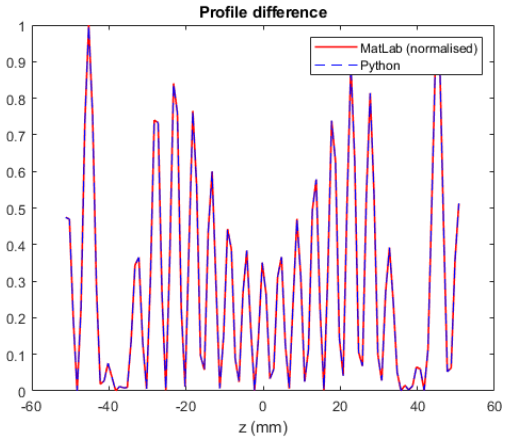


Figure 18: The normalized profile curves of the plots show in figure 15 with MATLAB in red and python in blue.

4.2.2 Simulation results (Python)

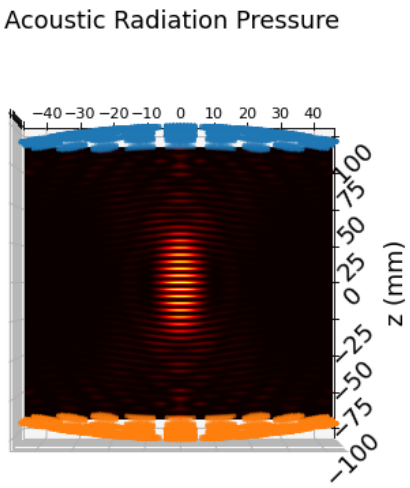


Figure 19: Acoustic radiation pressure simulated inside a MATBig, using the python-implementation of the matrix method.

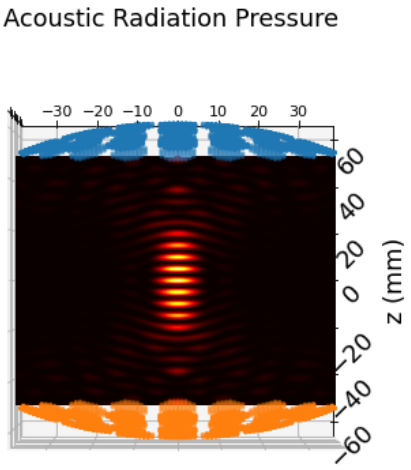


Figure 20: Acoustic radiation pressure simulated inside a MATMed, using the python-implementation of the matrix method.

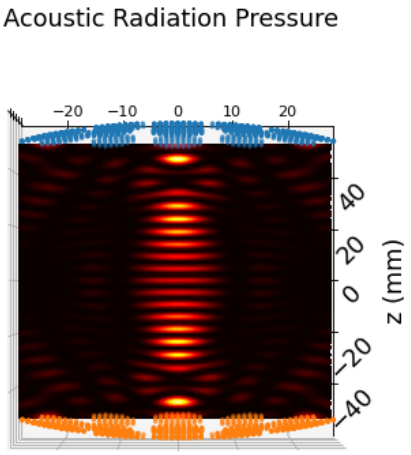


Figure 21: Acoustic radiation pressure simulated inside a MATSma, using the python-implementation of the matrix method.

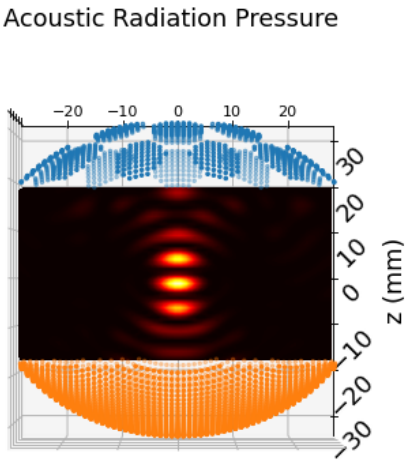


Figure 22: Result of simulation of a novel device using with an array of transducers, and a concave reflector, nicknamed "MicroLev".

4.3 Schlieren imaging

In this section the images taken of the acoustic potential using the schlieren imaging technique, for 4 different systems are presented.

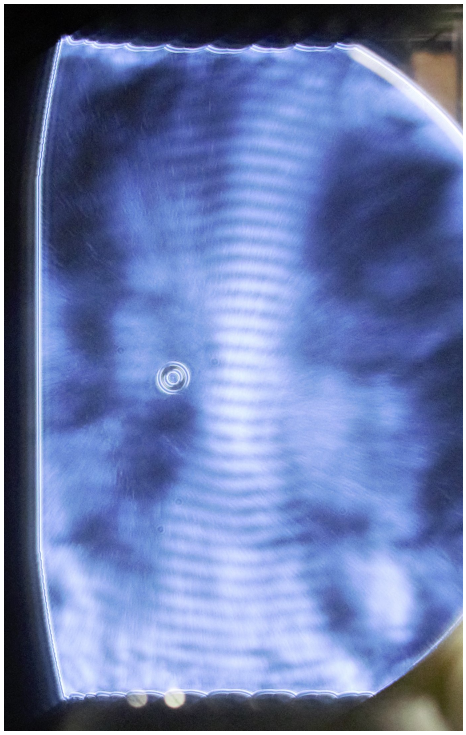


Figure 23: Schliere captured of the MATBig system.

Figure 24: Schliere captured of the MATMed system.

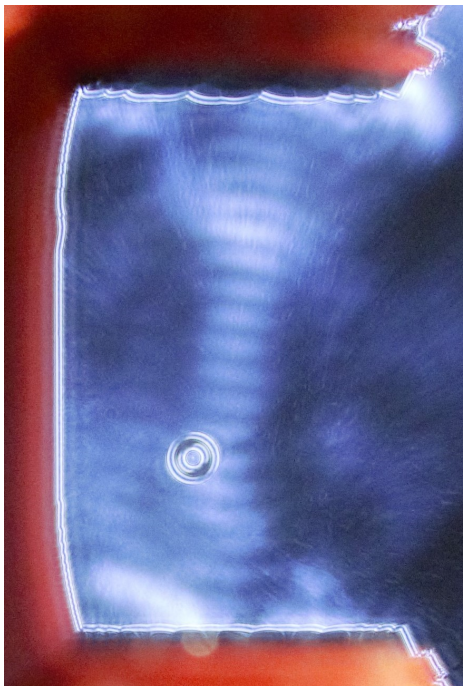


Figure 25: Schliere captured of the MATSma system.

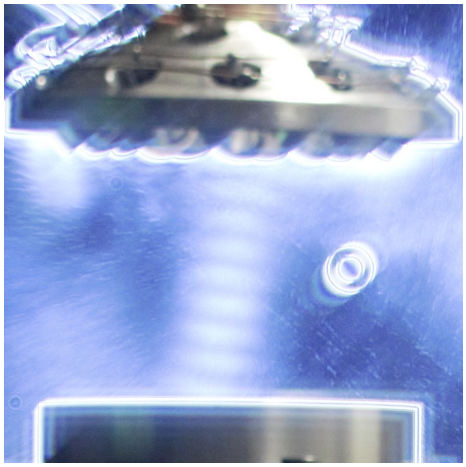


Figure 26: Schliere captured of the MicroLev system, a system using 17 transducers and a concave reflector.

4.4 Graphical User Interface

This section presents the GUI which wraps the python code that evaluates the matrix method for a given acoustic levitator system. The parameters of the system are defined by user input in the GUI and this section will explain how the system is defined by those parameters. The application has been named Simulation platform for Acoustic Levitation Traps (SALT) and requires python 3.7 to be installed on a users computer and the code is avaliable to be downloaded here. The user interface was built with a package called called `tkinter`, which comes pre-packaged with python. All plots are made with the `pyplot` module which is included in the `matplotlib` package and the mathetmatics of the matrix method are carried out with the `numpy` package. Both `numpy` and `matplotlib` need to be installed to run the application, if they are not installed when the program is run the user will be prompted to installed them.

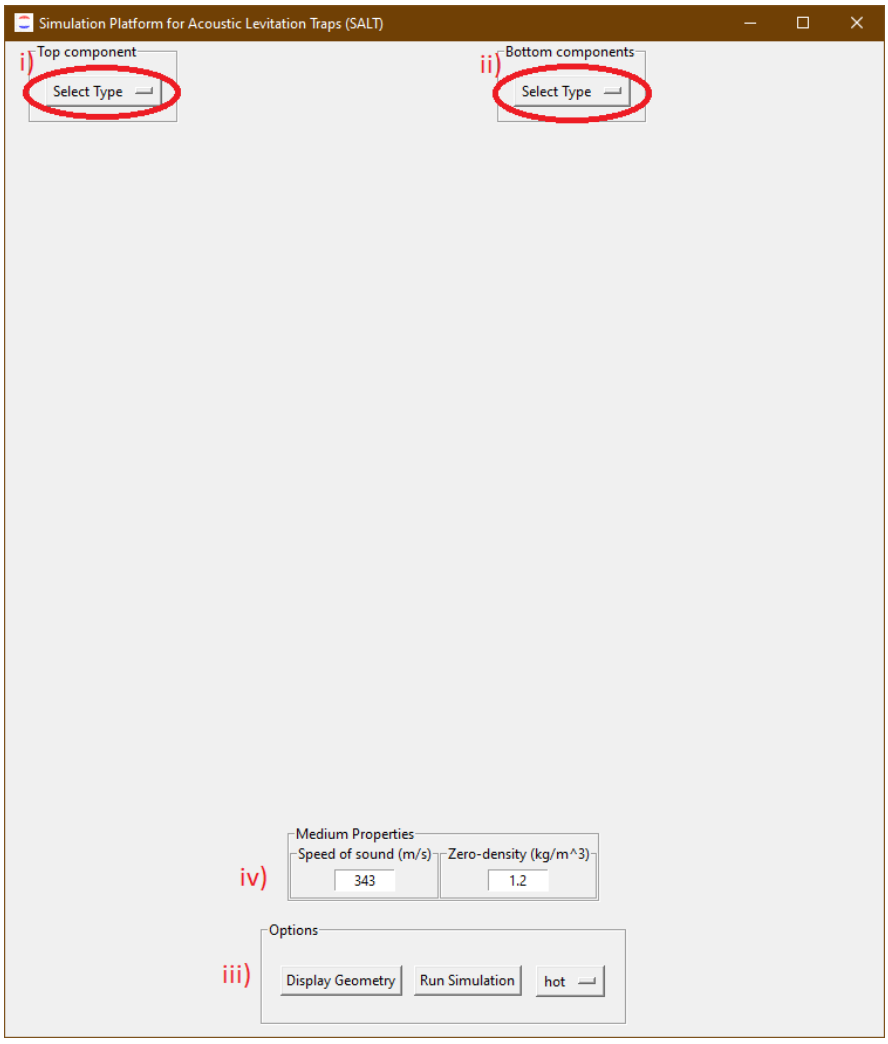


Figure 27: Upon launching the application, the user needs to select types for the top and bottom components of the acoustic levitation device from the drop-down menus indicated by i) and ii). At i) the options are *array* and *transducer*. The options for the bottom component are *array* and *reflector*. If none or only one of the components are selected before clicking either of the buttons at the bottom of the panel at iii), the script will show an error message in the console. The acoustic potential can also be simulated in mediums with different characteristics, that is the user can set by changing the speed of sound and medium density at iv). Normally the speed of sound is proportional to the inverse square-root of the density by the Newton-Laplace equation but in the matrix method density and speed of sound appear independent.

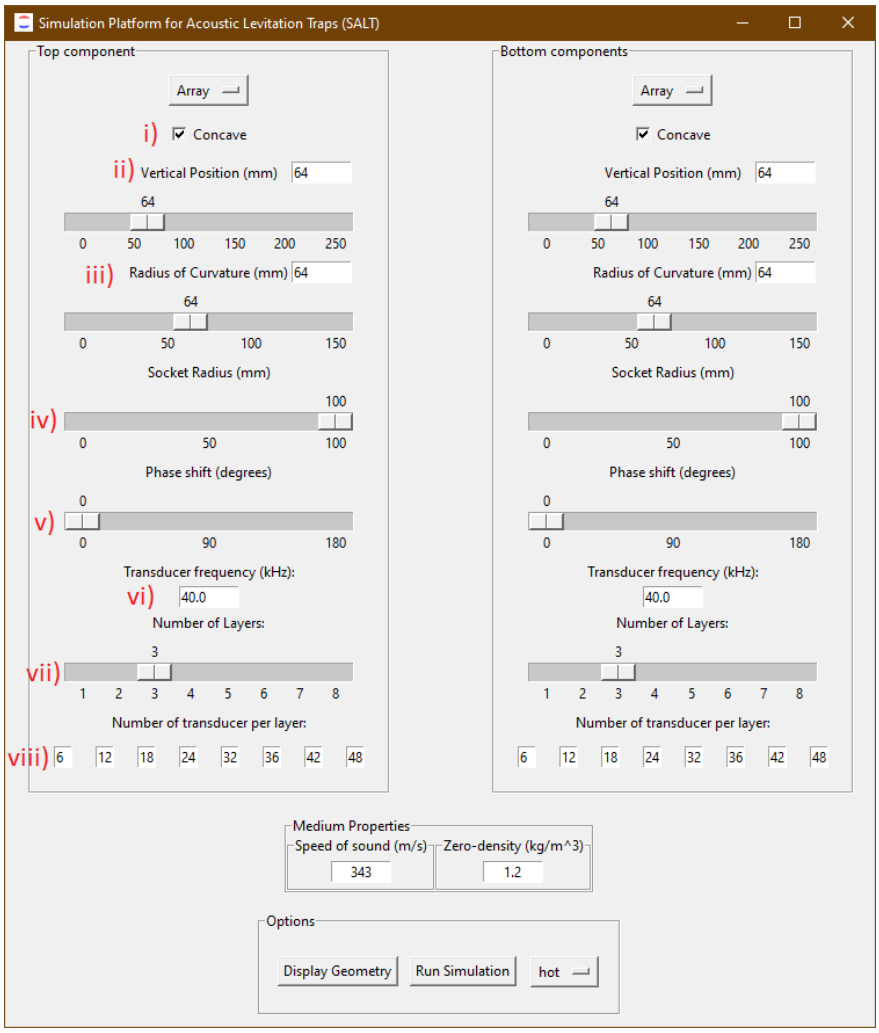


Figure 28: Here is an example of the application when a user has selected the option "array" for both top and bottom components. The significance of each parameter is listed in table 2.

Parameter	Significance
i)	Wether the array, reflector or transducer is flat or concave. Default setting is concave.
ii)	The distance from the "pit" of the array to the origin. The cavity length of the levitator is the sum of the vertical position of both components.
iii)	Radius of curvature determines the "steepness" of the bend of the array.
iv)	Socket radius is the size of the set of points which are generated when creating the array. This parameter is not really relevant for the size of the array as the radius of each ring is fixed but it does matter for other reasons which will be discussed in the next section.
v)	Phase shift of the emitted wave.
vi)	Frequency of the emitted wave.
vii)	Number of layers or rings on the array.
viii)	How many transducers should be in each ring or layer.

Table 2: Summary of the parameters marked in figure 28.

5 Discussion

In this section we provide a discussion on the results of the simulations and the limitations of the software.

5.1 Matrix Method Implementation in MATLAB and Python

The first implementation of the matrix method for this work was done in MATLAB in novemeber 2019. At that stage we had access to previous simulation work done on conventional acoustic levitaiton systems. Once we could conclude that our results were in good agreement with the previous work of Andrade [3], Stindt [7] and Baer [13], the development of a new implementation of the matrix method could proceed. The objective of the new implementation was to simulate the acoustic potential in a TinyLev-system. The biggest challenge with simulating the TinyLev system was to find a general method to generate the geometry of an array of transducers with dependence on the radius of curvature. This was eventually achieved by using conditional statements on points on a flat disk with a separation of 1mm. To add the radius of curvature the points were subsequently translated in the Z-axis. This method could be improved somewhat because the transducers aren't all represented by the same number of points which makes simulating a different phase for each transducer more difficult than it could be.

Subsequently, when the matrix method implementation for the TinyLev system was completed in MATLAB we proceeded to convert the code into python and build the GUI. The choice to use sliders to set the parameters was taken initially to make the application user-friendly but they can only take integer precision input and should be changed to text-based input which would allow for floating point precision.

5.2 Validation

To confirm that the Python implementation produces the same result as the one in MATLAB, we compare the acoustic radiation pressure visually side by side in figures 13-15.

Furthermore, we've plotted the normalized profile curves in figures 16-18. The profiles are taken from the middle column of values plotted in the previous figures and overlayed. By visual inspection it is obvious that the Python and MATLAB implementation generate identical results.

To validate the simulations qualitatively schlieren imaging of the MATBig, MATMed, MATSma and MicroLev systems were taken. The point of comparison with the schlieren images is the number of dark streaks in the schlieren image with the number of dark streaks in the surface-plot of the acoustic radiation pressure.

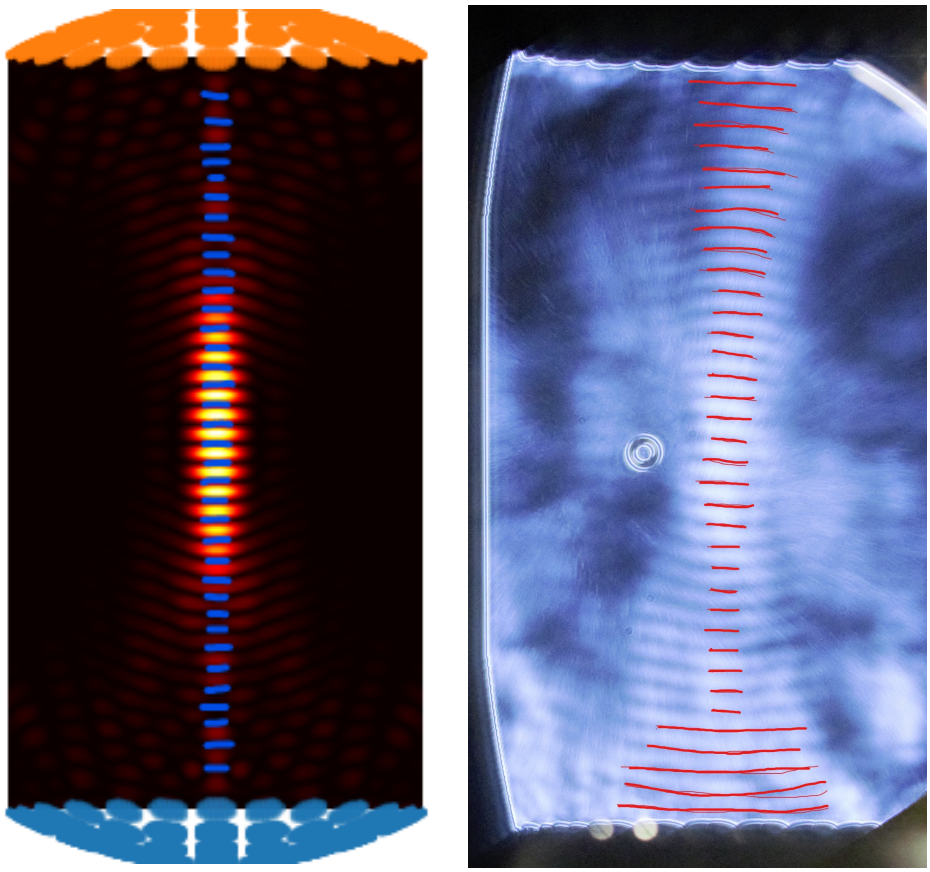


Figure 29: Schlieren image of the MATBig system with 36 dark streaks highlighted in red (right) and simulation of the MATBig systems with 36 dark streaks highlighted in blue (left).

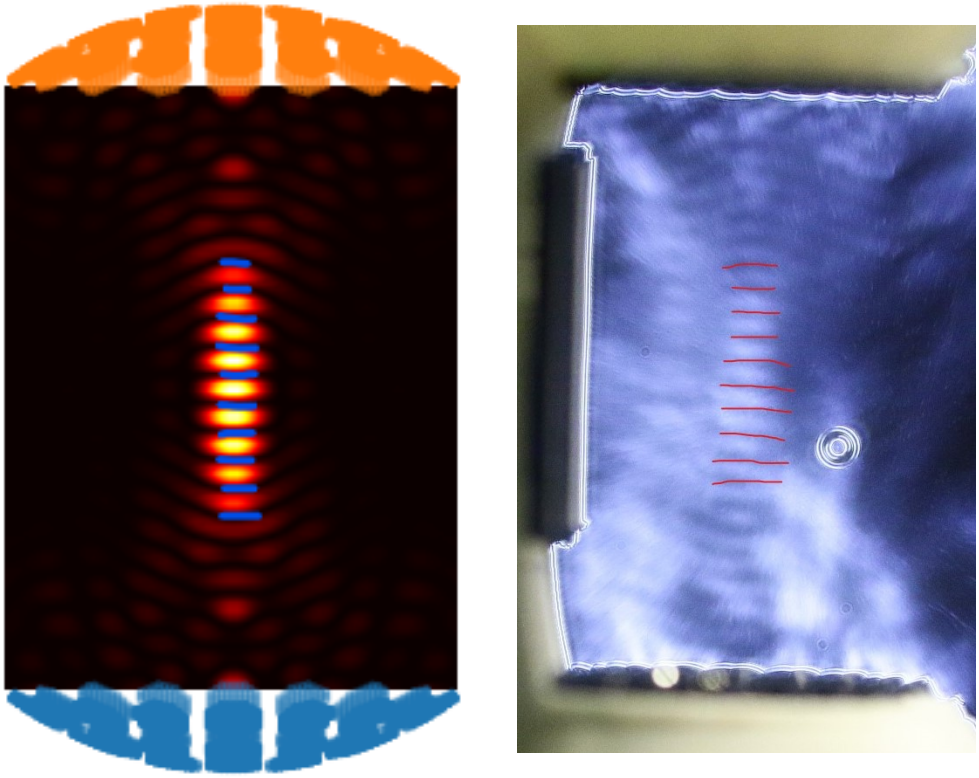


Figure 30: Schlieren image of the MATMed system with 10 dark streaks highlighted in red (right) and simulation of the MATMed system with 10 dark streaks highlighted in blue (left).

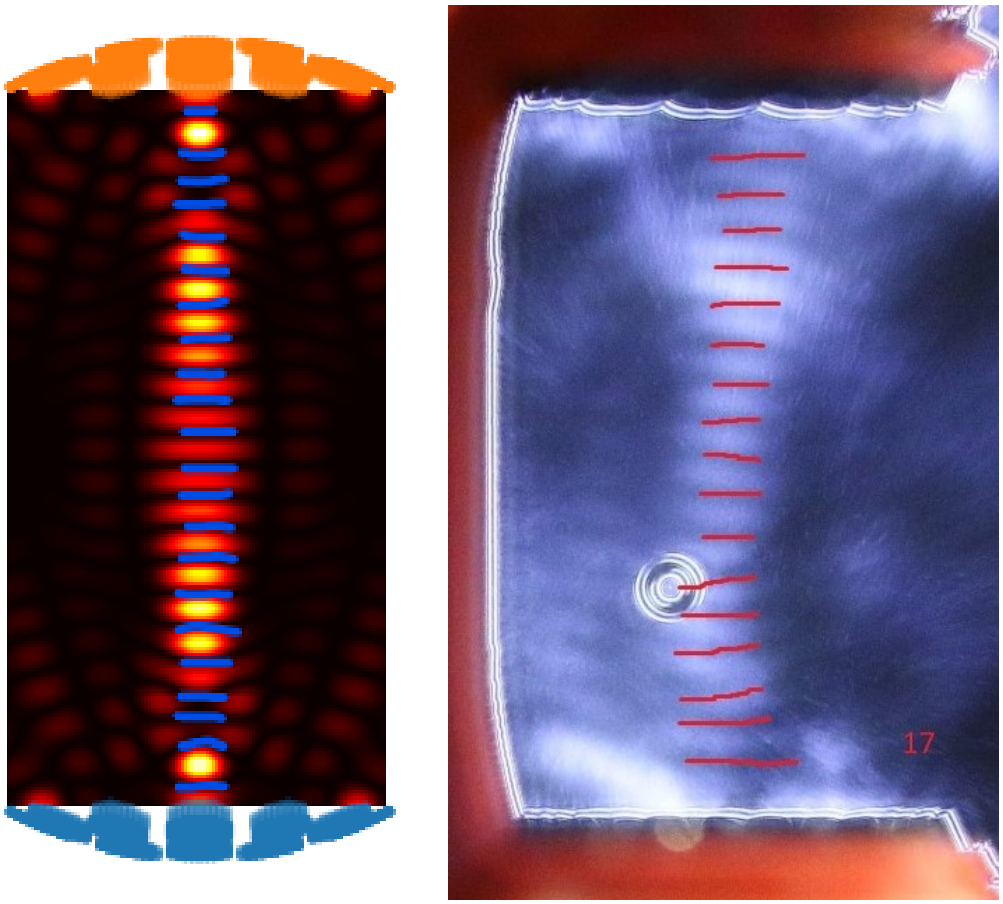


Figure 31: Schlieren image of the MATSma system with 17 dark streaks highlighted in red (right) and simulation of the MATSma system with 22 dark streaks highlighted in blue (left).

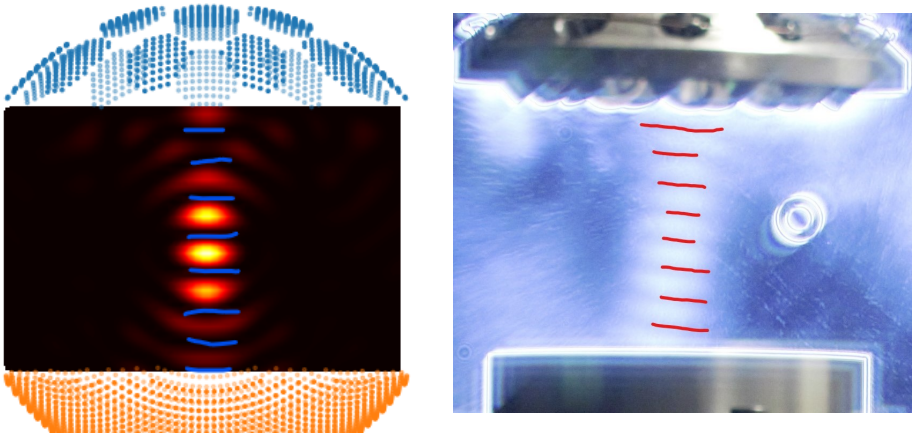


Figure 32: Schlieren image of the MicroLev system with 8 dark streaks highlighted in red (right) and simulation of the MicroLev system with 8 dark streaks highlighted in blue (left).

5.3 Error factors in schlieren imaging

There were some issues when recording the schlieren images, especially with the MATMed system, where heat radiating from the transducer array led to an obstructing effect in the images. To increase contrast of the schliere the voltage was increased from 12V to 20V which gave higher contrast at the central nodes but also increased the obstructing effect of heat near the surface of the transducers. In the MATMed system, only the 10 most central streaks have been marked because they were the strongest. The other lines seem to blend in with the background and couldn't be identified with any confidence. Another factor to consider when inspecting the schlieren images is that there are ineviteably imperfections in the structural components aswell as transducers used to build these devices. Such imperfections likely contribute further to breaking down the complex pattern of weaker nodes close to the arrays. There are especially many such weak nodes in the MATMed system, shown in figure 33.

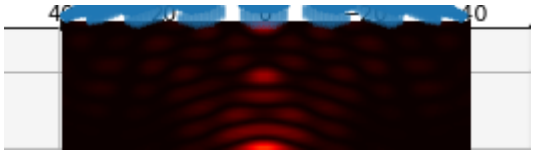


Figure 33: Zoom in on the close vicinity of the upper array in the MATMed system shows a complex pattern of weak nodes.

The effect of heat radiating from the arrays is not as obstructing in the MAT-Big system which may be caused by the much higher number of transducers,

where more transducers leads to a lower voltage per transducer and therefore less energy being converted to heat. A way that this might be mitigated is to optimise the schlieren photography with respect to levitator size and exciting voltage.

According to simulations, small changes in the cavity length influences the acoustic potential greatly. This can be seen by taking the value of the acoustic radiation pressure at the central antinode and varying the cavity length for a fixed radius of curvature. This is shown in figures 34 - 36. These differences could be present in the devices that have been used in this report due to (small) undocumented construction errors, affecting the outcome of the schlieren imaging-process. As for the physical MicroLev system which was used in to take schliere photography of, the measurement of the cavity length is only precise within 2-4 mm.

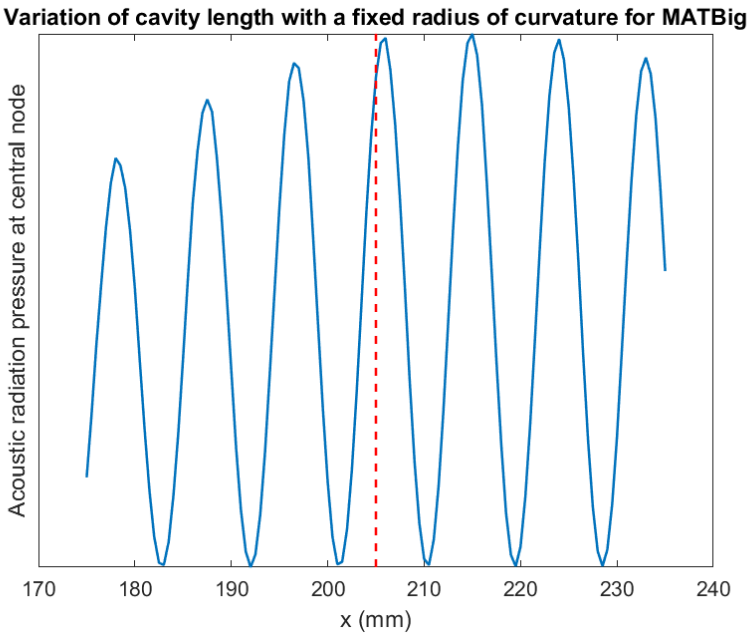


Figure 34: Demonstrating the effect of varying the cavity length of the MATBig system, according to the matrix method. The dashed red line indicates the cavity length used in the simulations and in the MATBig system.

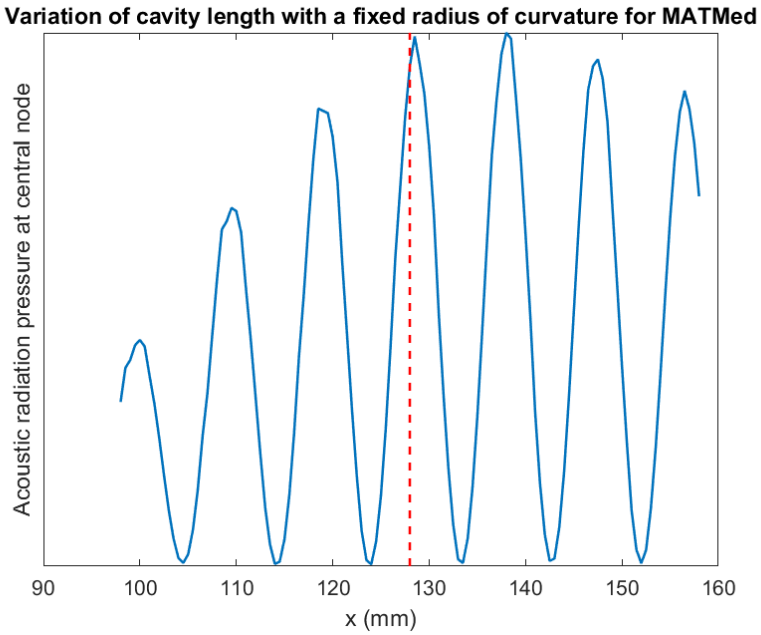


Figure 35: Demonstrating the effect of varying the cavity length of the MATMed system, according to the matrix method. The dashed red line indicates the cavity length used in the simulations and in the MATMed system.

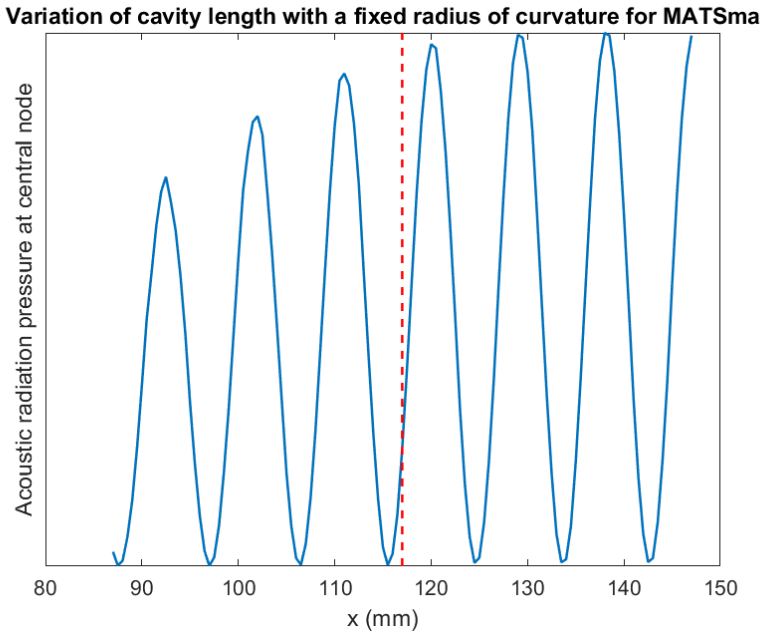


Figure 36: Demonstrating the effect of varying the cavity length of the MATSma system, according to the matrix method. The dashed red line indicates the cavity length used in the simulations and in the MATSma system.

The simulations suggest that the MATSma is in an unstable area where the acoustic radiation pressure at the node w.r.t cavity length is nearing an interregnum between being in and out of phase. In this interregnum the character of the acoustic trap changes as the waves coming from the two arrays go out of phase with each other. Such a shift can lead to increase or decrease in the number of visible nodes. This effect might be more complex than what the simulation can accurately depict since the matrix method really only solves for a first order perturbation term. The unstable area of the acoustic pressure w.r.t the cavity length may be why we see such a big qualitative difference (in the number of visible nodes) between the simulation of the MATSma system and its schlieren image. MATMed and MATBig are closer to the peaks of the curve.

Last, it should also be considered that the schlieren sensitivity to changes in air density is very sensitive to the relation between the size of the point light source and the light stop. In this work a very crude wire with a large area was used but since the wire is cylindrical, still some light can pass around it.

5.4 Poor alignment in the MicroLev system

The MicroLev system was poorly aligned due to the ad-hoc solution for mounting the reflector and array. This likely contributed to the "boxy" shape of the antinodes seen in the schlieren imaging, compared to the overall oval-shaped arrangement of schliere in the MATMed and MATBig as seen in figures 29 and 30. An example of schlieren imaging in a MicroLev system with better alignment was given by Victor Contreras, see figure 37.

Figure 37: An image of the MicroLev system, recorded and provided by Victor Contreras. The nodes take on an oval shape when the system is optimally aligned.

5.5 Bad geometry

When using the application to simulate the acoustic radiation pressure in an acoustic levitation device it is important to note that some settings will give bad results. The main parameter that users need to be aware of setting to a "good" value is the radius of curvature, as setting it too steep will create badly skewed transducer representations. Another critical setting is the disk radius,

iv) in figure 28. If this setting is too small, it will crop the array to a square and some layers of transducers will not be included, examples of bad geometries are shown in figures 38 and 39. Because of these artifacts it is important to use the "Display Geometry" option before running the simulation.

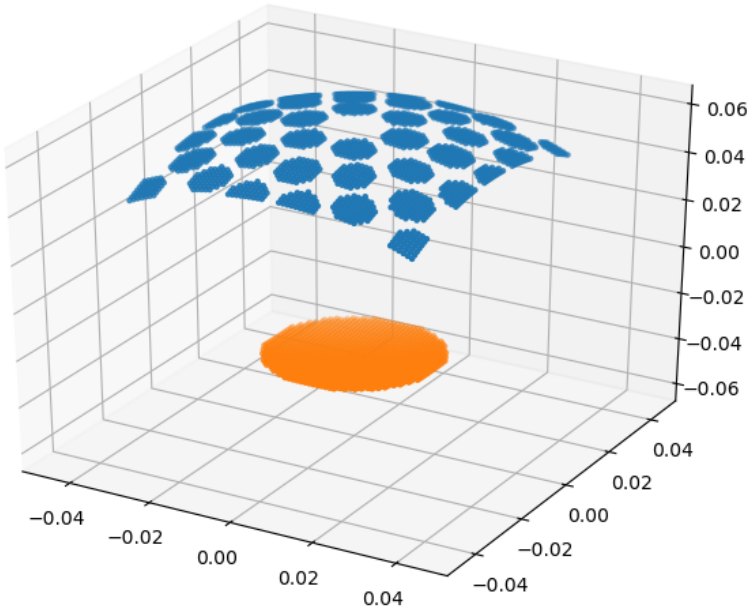


Figure 38: Example of when the radius of the array is set too low, resulting in the array being cropped to a square.

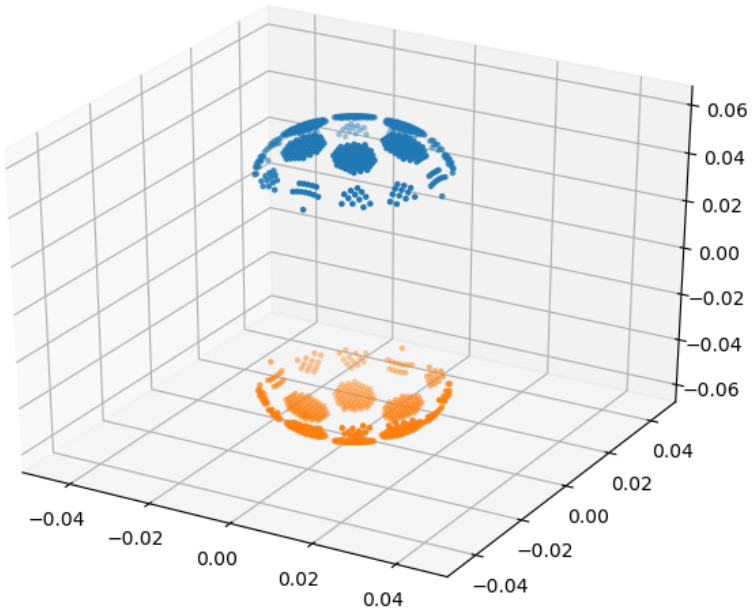


Figure 39: Example when the radius of curvature is set too low, resulting in cropping and skewing of the array.

6 Conclusion and outlook

It is safe to conclude that the MATLAB and python implementations qualitatively produce the same results. The overall quality of schlieren images should be considered fair but could be improved substantially. In the case of the MATBig and MicroLev systems our simulation produced the same number of dark streaks as the schlieren images showed which is considered very good agreement.

As the simulation could accurately show the 10 streaks of the MATMed system with the highest contrast in the central part of the cavity the simulation appears to be in good agreement with the qualitative validation. Since the remaining nodes are very weak (see figure 17) it is not surprising that they could not be detected using the schlieren technique.

As for the MATSma, there is some disagreement between the simulation and the number of nodes found in the schlieren images - these differences could be explained by the unstable neighbourhood w.r.t the cavity length which is predicted by the simulation in 36. It is left to future works to validate figures 34 to 36.

Lastly, it should be noted that the algorithm may scale very badly on computers with low RAM avaliability as large systems, mainly using reflectors and/or large transducers, generate a large number of points that takes up a

lot of RAM.

6.1 Future Work

Some suggestions for future work are listed below and will be explained further on.

1. Improve schlieren set up.
2. Validate figures 34 to 36.
3. Quantitative validation using rainbow schliere or microphone.
4. Fork the simulation application (SALT)

1) The schlieren setup could potentially be improved by using a plane black band instead of a wire which was used here. Using a plane black band could decrease the amount of unperturbed light that passes the light stop and therefore increase contrast of the dark and light streaks.
2) Figures 34 to 36 could be validated using using the schileren effect, as we have done here but with the purpose of investigating the interregnum between the peaks in the aforementioned figures.
3) It would be of great value to quantify the acoustic potential of these acoustic levitation devices and investigate the agreement of experimental data with simulations using the matrix method. Such experiments could be carried out either by using an unobtrusive microphone to measure the acoustic frequency inside the cavity. It could also hypothetically be carried out using rainbow schlieren which is a version of schlieren imaging where the light stop is replaced by a transparent film with a fine color gradient printed in a micro-meter sized band. Since the deflection of light in the schlieren effect is proportional to the density gradient of the area where the light passes, the pressure can be computed in each node by measuring the hue in the schlieren pictures.
4) Lastly, as with all software projects, their life can be extended and their characters altered by forking - students and seasoned programmers alike can take it upon themselves to make changes to the existing repository or clone it to make a version of their own. The most pressing improvements to be made would be to improve the way that the application handles errors and warnings and make sure that there are no bugs which break the application.

References

[1] A. Marzo, A. Barnes, and B. W. Drinkwater, "Tinylev: a multi-emitter single-axis acoustic levitator", Review of Scientific Instruments **88**, 085105 (2017).

[2] A. Marzo, T. Corkett, and B. W. Drinkwater, "Ultraino: an open phased-array system for narrowband airborne ultrasound transmission", IEEE transactions on ultrasonics, ferroelectrics, and frequency control **65**, 102–111 (2017).

[3]M. A. B. Andrade, N. Perez, F. Buiochi, and J. C. Adamowski, "Matrix method for acoustic levitation simulation", IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control **58**, 1674–1683 (2011).

[4]A. Ibáñez, C. Fritsch, M. Parrilla, and J. Villazón, "Monochromatic transfer matrix method for acoustic field simulation thorough media boundaries", Physics Procedia **3**, 883–890 (2010).

[5]T. Kozuka, K. Yasui, T. Tuziuti, A. Towata, and Y. Iida, "Acoustic standing-wave field for manipulation in air", Japanese Journal of Applied Physics **47**, 4336 (2008).

[6]T. Kozuka, K. Yasui, T. Tuziuti, A. Towata, J. Lee, and Y. Iida, "Measurement and numerical calculation of force on a particle in a strong acoustic field required for levitation", Japanese Journal of Applied Physics **48**, 07GM09 (2009).

[7]A. Stindt, M. Andrade, M. Albrecht, J. Adamowski, U. Panne, and J. Riedel, "Experimental and numerical characterization of the sound pressure in standing wave acoustic levitators", Review of Scientific Instruments **85**, 015110 (2014).

[8]M. A. B. Andrade, N. Pérez, and J. C. Adamowski, "Review of progress in acoustic levitation", Brazilian Journal of Physics **48**, 190–213 (2018).

[9]G. P. Thomas, M. A. Andrade, J. C. Adamowski, and E. C. Silva, "Acoustic levitation transportation of small objects using a ring-type vibrator", Physics Procedia **70**, 59–62 (2015).

[10]M. A. Andrade, F. T. Okina, A. L. Bernassau, and J. C. Adamowski, "Acoustic levitation of an object larger than the acoustic wavelength", The Journal of the Acoustical Society of America **141**, 4148–4154 (2017).

[11]M. A. Andrade, A. L. Bernassau, and J. C. Adamowski, "Acoustic levitation of a large solid sphere", Applied Physics Letters **109**, 044101 (2016).

[12]D. Sukhanov, S. Roslyakov, and F. Emelyanov, "Layer-by-layer application of particles using acoustic levitation", in Journal of physics: conference series, Vol. 1499, 1 (IOP Publishing, 2020), p. 012024.

[13]S. Baer, M. A. Andrade, C. Esen, J. C. Adamowski, G. Schweiger, and A. Ostendorf, "Analysis of the particle stability in a new designed ultrasonic levitation device", Review of Scientific Instruments **82**, 105111 (2011).

[14]A. Kundt, "Ueber eine neue art akustischer staubfiguren und über die anwendung derselben zur bestimmung der schallgeschwindigkeit in festen körpern und gasen", Annalen der Physik **203**, 497–523 (1866).

[15]K. Bücks and H. Müller, "Über einige beobachtungen an schwingenden piezoquarzen und ihrem schallfeld", Zeitschrift für Physik **84**, 75–86 (1933).

[16]F. Priego-Capote and L. de Castro, "Ultrasound-assisted levitation: lab-on-a-drop", TrAC Trends in Analytical Chemistry **25**, 856–867 (2006).

[17]W. Xie and B. Wei, "Dependence of acoustic levitation capabilities on geometric parameters", Physical Review E **66**, 026605 (2002).

[18]W. Xie and B. Wei, "Parametric study of single-axis acoustic levitation", Applied Physics Letters **79**, 881–883 (2001).

[19]A. Yarin, M. Pfaffenlehner, and C. Tropea, "On the acoustic levitation of droplets", Journal of Fluid Mechanics **356**, 65–91 (1998).

[20]C. Shen, W. Xie, and B. Wei, "Parametrically excited sectorial oscillation of liquid drops floating in ultrasound", Physical Review E **81**, 046305 (2010).

[21]V. Contreras, R. Valencia, J. Peralta, H. Sobral, M. A. Meneses-Nava, and H. Martinez, "Chemical elemental analysis of single acoustic-levitated water droplets by laser-induced breakdown spectroscopy", Opt. Lett. **43**, 2260–2263 (2018).

[22]V. Contreras, A. Díaz, S. Tan, and H. Martínez, "Laser induced breakdown spectroscopy on liquids assisted by acoustic levitation sampling: towards on-line monitoring applications", in Latin america optics and photonics conference (2018), Tu3E.5.

[23]S. Santesson and S. Nilsson, "Airborne chemistry: acoustic levitation in chemical analysis", Analytical and bioanalytical chemistry **378**, 1704–1709 (2004).

[24]M. Meneses-Nava, I. Rosas-Roman, O. Barbosa-Garcıa, M. Rodriguez, and J. Maldonado, "Stability evaluation of water droplets levitated by a tinylev acoustic levitator for laser induced breakdown spectroscopy", Spectrochimica Acta Part B: Atomic Spectroscopy, 105855 (2020).

[25]L. E. Kinsler, A. R. Frey, A. B. Coppens, and J. V. Sanders, *Fundamentals of acoustics* (1999).

[26]R. T. Beyer, "Radiation pressure—the history of a mislabeled tensor", The Journal of the Acoustical Society of America **63**, 1025–1030 (1978).

[27]G. S. Settles and M. J. Hargather, "A review of recent developments in schlieren and shadowgraph techniques", Measurement Science and Technology **28**, 042001 (2017).

[28]L. P. Gor'kov, "On the forces acting on a small particle in an acoustical field in an ideal fluid", in Sov. phys. dokl. Vol. 6 (1962), pp. 773–775.

[29]T. G. Wang, "Acoustic levitation and manipulation for space applications", (1979).

[30]D. Haydock and J. Yeomans, "Lattice boltzmann simulations of acoustic streaming", Journal of Physics A: Mathematical and General **34**, 5201 (2001).

[31]D. Haydock, "Lattice boltzmann simulations of the time-averaged forces on a cylinder in a sound field", Journal of Physics A: Mathematical and General **38**, 3265 (2005).

[32]G. Barrios and R. Rechtman, "Dynamics of an acoustically levitated particle using the lattice boltzmann method", Journal of Fluid Mechanics **596**, 191–200 (2008).

[33]S. Chen and G. D. Doolen, "Lattice boltzmann method for fluid flows", Annual review of fluid mechanics **30**, 329–364 (1998).

[34]U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas automata for the navier-stokes equation", Physical review letters **56**, 1505 (1986).

[35]*7.1: cellular automata - the nature of code*, https://www.youtube.com/watch?v=DKGodqDs9sA, Accessed: 2020-10-27.

[36]*7.2: wolfram elementary cellular automata - the nature of code*, https://www.youtube.com/watch?v=W1zKu3fDQR8, Accessed: 2020-10-27.

[37]*Cellular automata and rule 30 (stephen wolfram) — ai podcast clips*, https://www.youtube.com/watch?v=VguG_y05Xe8, Accessed: 2020-10-27.

[38]*Von neumann neighborhood, wikipedia*, https://en.wikipedia.org/wiki/Von_Neumann_neighborhood, Accessed: 2020-10-27.

[39]P. Glynne-Jones, P. P. Mishra, R. J. Boltryk, and M. Hill, "Efficient finite element modeling of radiation forces on elastic particles of arbitrary size and geometry", The Journal of the Acoustical Society of America **133**, 1885–1893 (2013).

[40]Z. Hong, W. Zhai, N. Yan, and B. Wei, "Measurement and simulation of acoustic radiation force on a planar reflector", The Journal of the Acoustical Society of America **135**, 2553–2558 (2014).

[41]M. A. Andrade, F. Buiochi, and J. C. Adamowski, "Finite element analysis and optimization of a single-axis acoustic levitator", IEEE transactions on ultrasonics, ferroelectrics, and frequency control **57**, 469–479 (2010).

[42]*Acoustic levitator instructable*, https://www.instructables.com/id/Acoustic-Levitator/, Accessed: 2020-07-21.

[43]J. Jellstam, "Acoustic levitation as mobile visual representation", MA thesis (University of Gothenburg, Gothenburg, June 2020).

A Code

A.1 Main file

```
from gui import masterframe

root = masterframe()
root.setouterproperties()
root.placemainwidgets()
root.mainloop()
```

A.2 GUI

```
import tkinter as tk
import time, sys, os

try:
    import numpy as np
except:
    os.system("pip install numpy")
    import numpy as np

try:
    from matplotlib import pyplot as plt
except:
    os.system("pip install matplotlib")
    from matplotlib import pyplot as plt

from MatrixMethod import MatrixMethod, CreateGeometry
from mpl_toolkits.mplot3d import axes3d, Axes3D

def definedicts(page_inst, masterframe_inst):
    c = float(masterframe_inst.medium_c.get())
    rho = float(masterframe_inst.medium_density.get())

    medium_properties = {"Density": rho, "SpeedOfSound": c}

    top_properties = {"Orientation":1,
    "Type": masterframe_inst.top_selection.get(),
    "Concave":page_inst.properties_page1["Concave"].get()}

    bot_properties = {"Orientation":-1,
    "Type":masterframe_inst.bot_selection.get(),
    "Concave":page_inst.properties_page2["Concave"].get()}
```

```
    for key in page_inst.properties_page1:
        if key != "Depth":
            top_properties.update({key : page_inst.properties_page1[key].get()})
            if key in ["Displacement", "RadiusCurvature", "Radius"]:
                top_properties[key] = float(top_properties[key])*1e-3
            if key == "TransFreq":
                top_properties[key] = float(top_properties[key])*1e3
            if key == "Layers":
                top_properties[key] = int(top_properties[key])
            if key == "Phase":
                top_properties[key] = int(top_properties[key])

        else:
            v = []
            for item in page_inst.properties_page1[key]:
                v = np.append(v, int(item.get()))
            top_properties.update({key : v})

    for key in page_inst.properties_page2:
        if key != "Depth":
            bot_properties.update({key : page_inst.properties_page2[key].get()})
            if key in ["Displacement", "RadiusCurvature", "Radius"]:
                try:
                    bot_properties[key] = float(bot_properties[key])*1e-3
                except ValueError:
                    print("Error with key: ", key)
            if key == "TransFreq":
                bot_properties[key] = float(bot_properties[key])*1e3
            if key == "Layers":
                bot_properties[key] = int(bot_properties[key])
            if key == "Phase":
                bot_properties[key] = int(bot_properties[key])
        else:
            v = []
            for item in page_inst.properties_page2[key]:
                v = np.append(v, int(item.get()))
            bot_properties.update({key : v})

    return bot_properties, top_properties, medium_properties

class types:
    def array(self, frame):
        """ Method adds widgets relevant to an array of transducers """

        labels, scales, entries = [], [], []
        properties = {}
```

```
var_checkbox = tk.BooleanVar()
checkbox = tk.Checkbutton(frame, text="Concave", variable=var_checkbox,
onvalue=True, offvalue=False)
checkbox.pack(pady=10)
var_checkbox.set(True)
properties.update({"Concave" : var_checkbox})

labels = np.append(labels, tk.Label(frame, text="Vertical Position (mm)"))
labels[0].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=250, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[0].pack(padx=10, pady=2)
# scales[0].set(68)

entries = np.append(entries, tk.Entry(frame, width = 8,
xscrollcommand = lambda x, y: scales[0].set(float(entries[0].get())) ))
entries[0].pack(padx=2, pady=2)
entries[0].place(x=215, y=77)
entries[0].insert(0, 64)

properties.update({"Displacement" : entries[0]})

labels = np.append(labels, tk.Label(frame, text="Radius of Curvature (mm)"))
labels[1].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=150, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[1].pack(padx=10, pady=2)
scales[1].set(64)

entries = np.append(entries, tk.Entry(frame, width = 8,
xscrollcommand = lambda x, y: scales[1].set(float(entries[1].get())) ))
entries[1].pack(padx=2, pady=2)
entries[1].place(x=215, y=163)
entries[1].insert(0, 64)

properties.update({"RadiusCurvature" : entries[1]})

labels = np.append(labels, tk.Label(frame, text="Socket Radius (mm)"))
labels[2].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=100, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[2].pack(padx=10, pady=2)
scales[2].set(100)

properties.update({"Radius" : scales[2]})
```

```
        labels = np.append(labels, tk.Label(frame, text="Phase shift (degrees)"))
        labels[3].pack(pady=0)
        scales = np.append(scales, tk.Scale(frame, from_=0, to=180, tickinterval=90,
        orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
        scales[3].pack(padx=10, pady=2)

        properties.update({"Phase" : scales[3]})

        labels = np.append(labels, tk.Label(frame, text="Transducer frequency (kHz):"))
        labels[4].pack(pady=0)
        entries = np.append(entries, tk.Entry(frame, width = 8))
        entries[2].pack(padx=10, pady=2)
        entries[2].insert(0, 40.0)

        properties.update({"TransFreq" : entries[2]})

        labels = np.append(labels, tk.Label(frame, text="Number of Layers:"))
        labels[5].pack(pady=0)
        scales = np.append(scales, tk.Scale(frame, from_=1, to=8, tickinterval=1,
        orient=tk.HORIZONTAL, variable = tk.IntVar(), length=250))
        scales[4].pack(padx=10, pady=2)
        scales[4].set(3)

        properties.update({"Layers" : scales[4]})

        labels = np.append(labels, tk.Label(frame,
        text = "Number of transducer per layer:"))
        labels[6].pack(pady=0)
        v = [6, 12, 18, 24, 32, 36, 42, 48]
        for i in range(0,8):
            entries = np.append(entries, tk.Entry(frame, width=2))
            entries[3+i].pack(side=tk.LEFT, padx=10, pady=10)
            entries[3+i].insert(0,v[i])

        properties.update({"Depth" : entries[3:]})

        return properties, labels, scales, entries, checkbox

    def transducer(self, frame):
        """ Method adds widgets relevant to a transducer """
        labels, scales, entries = [], [], []
        properties = {}

        var_checkbox = tk.BooleanVar()
        checkbox = tk.Checkbutton(frame, text="Concave", variable=var_checkbox,
```

```
onvalue=True, offvalue=False)
checkbox.pack(pady=10)

properties.update({"Concave" : var_checkbox})

labels = np.append(labels, tk.Label(frame, text="Vertical Position (mm)"))
labels[0].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=250, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[0].pack(padx=10, pady=2)
# scales[0].set(68)

entries = np.append(entries, tk.Entry(frame, width = 8,
xscrollcommand = lambda x, y: scales[0].set(float(entries[0].get())) ))
entries[0].pack(padx=2, pady=2)
entries[0].place(x=215, y=77)
entries[0].insert(0, 68.6)

properties.update({"Displacement" : entries[0]})

labels = np.append(labels, tk.Label(frame, text="Radius of Curvature (mm)"))
labels[1].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=150, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[1].pack(padx=10, pady=2)
scales[1].set(64)

entries = np.append(entries, tk.Entry(frame, width = 8,
xscrollcommand = lambda x, y: scales[1].set(float(entries[1].get())) ))
entries[1].pack(padx=2, pady=2)
entries[1].place(x=215, y=163)
entries[1].insert(0, 64)

properties.update({"RadiusCurvature" : entries[1]})


labels = np.append(labels, tk.Label(frame, text="Socket Radius (mm)"))
labels[2].pack(pady=0)
scales = np.append(scales, tk.Scale(frame, from_=0, to=100, tickinterval=50,
orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
scales[2].pack(padx=10, pady=2)
scales[2].set(35)

properties.update({"Radius" : scales[2]})

labels = np.append(labels, tk.Label(frame, text="Phase shift (degrees)"))
```

```
        labels[3].pack(pady=0)
        scales = np.append(scales, tk.Scale(frame, from_=0, to=180, tickinterval=90,
        orient=tk.HORIZONTAL, variable = tk.IntVar(), length=250))
        scales[3].pack(padx=10, pady=2)

        properties.update({"Phase" : scales[3]})

        labels = np.append(labels, tk.Label(frame, text="Transducer frequency (kHz):"))
        labels[4].pack(pady=0)
        entries = np.append(entries, tk.Entry(frame, width = 8))
        entries[1].pack(padx=10, pady=2)
        entries[1].insert(0,40.0)
        properties.update({"TransFreq" : entries[1]})

        return properties, labels, scales, entries, checkbox

    def reflector(self, frame):
        """ Method adds widgets relevant to a reflector """
        labels, scales, entries = [], [], []
        properties = {}

        var_checkbox = tk.BooleanVar()
        checkbox = tk.Checkbutton(frame, text="Concave", variable=var_checkbox,
        onvalue=True, offvalue=False)
        checkbox.pack(pady=10)
        var_checkbox.set(True)

        properties.update({"Concave" : var_checkbox})

        labels = np.append(labels, tk.Label(frame, text="Vertical Position (mm)"))
        labels[0].pack(pady=0)
        scales = np.append(scales, tk.Scale(frame, from_=0, to=250, tickinterval=50,
        orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
        scales[0].pack(padx=10, pady=2)
        # scales[0].set(68)

        entries = np.append(entries, tk.Entry(frame, width = 8,
        xscrollcommand = lambda x, y: scales[0].set(float(entries[0].get())) ))

        entries[0].pack(padx=2, pady=2)
        entries[0].place(x=215, y=77)
        entries[0].insert(0, 33)

        properties.update({"Displacement" : entries[0]})

        labels = np.append(labels, tk.Label(frame, text="Radius of Curvature (mm)"))
```

```
            labels[1].pack(pady=0)
            scales = np.append(scales, tk.Scale(frame, from_=0, to=150, tickinterval=50,
            orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
            scales[1].pack(padx=10, pady=2)
            scales[1].set(64)

            entries = np.append(entries, tk.Entry(frame, width = 8,
            xscrollcommand = lambda x, y: scales[1].set(float(entries[1].get())) ))
            entries[1].pack(padx=2, pady=2)
            entries[1].place(x=215, y=163)
            entries[1].insert(0, 64)

            properties.update({"RadiusCurvature" : entries[1]})

            labels = np.append(labels, tk.Label(frame, text="Socket Radius (mm)"))
            labels[2].pack(pady=0)
            scales = np.append(scales, tk.Scale(frame, from_=0, to=100, tickinterval=50,
            orient=tk.HORIZONTAL, variable = tk.DoubleVar(), length=250))
            scales[2].pack(padx=10, pady=2)
            scales[2].set(30)

            properties.update({"Radius" : scales[2]})

            return properties, labels, scales, entries, checkbox

    class Pager:
        def __init__(self):
            self.labels_page1 = []
            self.labels_page2 = []
            self.properties_page1 = []
            self.properties_page2 = []
            self.scales_page1 = []
            self.scales_page2 = []
            self.entries_page1 = []
            self.entries_page2 = []

        def toggle_page1(self, selection, frame):
            type = types()

            if selection.get() == "Transducer":
                if len(self.labels_page1) > 0:
                    for i in range(len(self.labels_page1)):
                        self.labels_page1[i].destroy()
                    for i in range(len(self.scales_page1)):
                        self.scales_page1[i].destroy()
                    for i in range(len(self.entries_page1)):
```

```
                self.entries_page1[i].destroy()
            self.checkbox_page1.destroy()

        self.properties_page1, self.labels_page1, self.scales_page1,
        self.entries_page1, self.checkbox_page1 = type.transducer(frame)

    if selection.get() == "Array":
        if len(self.labels_page1) > 0:
            for i in range(len(self.labels_page1)):
                self.labels_page1[i].destroy()
            for i in range(len(self.scales_page1)):
                self.scales_page1[i].destroy()
            for i in range(len(self.entries_page1)):
                self.entries_page1[i].destroy()
            self.checkbox_page1.destroy()

        self.properties_page1, self.labels_page1, self.scales_page1,
        self.entries_page1, self.checkbox_page1 = type.array(frame)

def toggle_page2(self, selection, frame):
    type = types()
    if selection.get() == "Reflector":
        if len(self.labels_page2) > 0:
            for i in range(len(self.labels_page2)):
                self.labels_page2[i].destroy()
            for i in range(len(self.scales_page2)):
                self.scales_page2[i].destroy()
            for i in range(len(self.entries_page2)):
                self.entries_page2[i].destroy()
            self.checkbox_page2.destroy()

        self.properties_page2, self.labels_page2, self.scales_page2,
        self.entries_page2, self.checkbox_page2 = type.reflector(frame)

    if selection.get() == "Array":
        if len(self.labels_page2) > 0:
            for i in range(len(self.labels_page2)):
                self.labels_page2[i].destroy()
            for i in range(len(self.scales_page2)):
                self.scales_page2[i].destroy()
            for i in range(len(self.entries_page2)):
                self.entries_page2[i].destroy()
            self.checkbox_page2.destroy()

        self.properties_page2, self.labels_page2, self.scales_page2,
        self.entries_page2, self.checkbox_page2 = type.array(frame)
```

```
def render(self, masterframe_inst):

    if masterframe_inst.top_selection.get() != "Select Type" or masterframe_inst.bot_sel

        bot_properties, top_properties,
        medium_properties = definedicts(self, masterframe_inst)

        Ux, Uy, Uz, Vx, Vy, Vz = CreateGeometry(top_properties, bot_properties)

        fig = plt.figure()
        ax = Axes3D(fig)
        ax.scatter3D(Vx, Vy, Vz, marker='.')
        ax.scatter3D(Ux, Uy, Uz, marker='.')
        ax.set_xlim([-0.05,0.05])
        ax.set_ylim([-0.05,0.05])
        ax.set_zlim([-0.065,0.065])
        plt.show()
    else:
        print("Please define system geometry")

def compute_potential(self, masterframe_inst):
    top_selection = masterframe_inst.top_selection
    bot_selection = masterframe_inst.bot_selection

    if top_selection.get() != "Select Type" or bot_selection.get() != "Select Type":
        bot_properties, top_properties,
        medium_properties = definedicts(self, masterframe_inst)

        Ux, Uy, Uz, Vx, Vy, Vz = CreateGeometry(top_properties, bot_properties)

        start = time.time()
        acoustic_radiation_pressure, relative_potential, pressure, x_span, z_span = Matr
        end = time.time()
        diff = end - start
        print("Total time elapsed was %.4f" % diff, "seconds")
        x = len(x_span)
        z = len(z_span)

        xmax=np.max(x_span)*1e3
        xmin=np.min(x_span)*1e3
        zmax=np.max(z_span)*1e3
        zmin=np.min(z_span)*1e3

        _relative_potential = np.real(relative_potential).reshape([z, x])
```

```
            X, Y = np.mgrid[zmin:zmax,xmin:xmax]
            Z = _relative_potential

            maxPotential = np.max(Z)
            minPotential = np.min(Z)

            colormap = masterframe_inst.colormap.get()

            fig = plt.figure(figsize=(8, 6), dpi = 80, facecolor='w', edgecolor='k')
            ax = fig.add_subplot(111, projection='3d')
            ax.scatter(Vz*1e3, Vy*1e3, Vx*1e3, marker='.')
            ax.scatter(Uz*1e3, Uy*1e3, Ux*1e3, marker='.')
            ax.set_xlim([-50,50])
            ax.set_ylim([-50,50])
            ax.set_zlim([-50,50])
            ax.view_init(azim=0,elev=90)

            levels = np.linspace(minPotential,maxPotential,1000)
            cset = ax.contourf(X, Y, Z, levels, cmap=colormap)
            ax.clabel(cset, fontsize=9, inline=1)
            ax.set_xlabel("z (mm)",fontsize=16, labelpad=16)
            plt.xticks(rotation=45)
            plt.title("Acoustic Radiation Pressure", fontsize=16)
            plt.xticks(size=16)
            plt.show()
        else:
            print("Please define system geometry")

    def compute_acoustic_rad(self, masterframe_inst):
        top_selection = masterframe_inst.top_selection
        bot_selection = masterframe_inst.bot_selection

        if top_selection.get() != "Select Type" or bot_selection.get() != "Select Type":
            bot_properties, top_properties,
            medium_properties = definedicts(self, masterframe_inst)

            Ux, Uy, Uz, Vx, Vy, Vz = CreateGeometry(top_properties, bot_properties)

            start = time.time()
            _acoustic_radiation_pressure, relative_potential,
            pressure, x_span, z_span = MatrixMethod(medium_properties,top_properties,bot_pro
            end = time.time()
            diff = end - start
            print("Total time elapsed was %.4f" % diff, "seconds")
            x_len = len(x_span)
```

```
z_len = len(z_span)

xmax=np.max(x_span)*1e3
xmin=np.min(x_span)*1e3
zmax=np.max(z_span)*1e3
zmin=np.min(z_span)*1e3

X, Y = np.mgrid[zmin:zmax,xmin:xmax]
Z = _acoustic_radiation_pressure

maxPotential = np.max(_acoustic_radiation_pressure)
minPotential = np.min(_acoustic_radiation_pressure)

colormap = masterframe_inst.colormap.get()

fig = plt.figure(figsize=(8, 6), dpi = 80, facecolor='w', edgecolor='k')
ax = fig.add_subplot(111, projection='3d')
ax.scatter(Vz*1e3, Vy*1e3, Vx*1e3, marker='.')
ax.scatter(Uz*1e3, Uy*1e3, Ux*1e3, marker='.')
ax.set_xlim([-50,50])
ax.set_ylim([-50,50])
ax.set_zlim([-50,50])
ax.view_init(azim=180,elev=90)

levels = np.linspace(minPotential,maxPotential,1000)
cset = ax.contourf(X, Y, Z, levels, cmap=colormap)
ax.clabel(cset, fontsize=9, inline=1)
ax.set_xlabel("z (mm)",fontsize=16, labelpad=16)
plt.xticks(rotation=45)
plt.title("Acoustic Radiation Pressure", fontsize=16)
plt.xticks(size=16)

middle_index = int(np.ceil(x_len/2))

profile = Z[:,middle_index]

fig = plt.figure(figsize=(8, 6), dpi = 80, facecolor='w', edgecolor='k')
plt.title("Radiation Pressure Profile", fontsize=16)
ax = plt.gca()

xPlot=np.linspace(zmin, zmax, z_len)
plt.plot(xPlot,profile)
ax.set_xticks(xPlot[::5])
ax.set_xlabel("z (mm)",fontsize=16, labelpad=5)
plt.xticks(rotation=45)
```

```
            np.savetxt('profile.txt', profile, delimiter=',')
            np.savetxt('xPlot.txt', xPlot, delimiter=',')

            plt.show()

        else:
            print("Please define system geometry")

    def compute_pressure(self, masterframe_inst):
        top_selection = masterframe_inst.top_selection
        bot_selection = masterframe_inst.bot_selection

        if top_selection.get() != "Select Type" or bot_selection.get() != "Select Type":
            bot_properties, top_properties,
            medium_properties = definedicts(self, masterframe_inst)

            Ux, Uy, Uz, Vx, Vy, Vz = CreateGeometry(top_properties, bot_properties)

            start = time.time()
            acoustic_radiation_pressure, relative_potential,
            pressure, x_span, z_span=
            MatrixMethod(medium_properties,top_properties,bot_properties)
            end = time.time()
            diff = end - start
            print("Total time elapsed was %.4f" % diff, "seconds")
            x = len(x_span)
            z = len(z_span)

            xmax=np.max(x_span)*1e3
            xmin=np.min(x_span)*1e3
            zmax=np.max(z_span)*1e3
            zmin=np.min(z_span)*1e3

            _pressure = np.real(pressure).reshape([z, x])
            maxPressure = np.max(_pressure)
            minPressure = np.min(_pressure)

            X, Y = np.mgrid[zmin:zmax,xmin:xmax]
            Z = _pressure

            maxPotential = np.max(Z)
            minPotential = np.min(Z)

            colormap = masterframe_inst.colormap.get()

            fig = plt.figure(figsize=(8, 6), dpi = 80, facecolor='w', edgecolor='k')
```

```
            ax = fig.add_subplot(111, projection='3d')
            ax.scatter(Vz*1e3, Vy*1e3, Vx*1e3, marker='.')
            ax.scatter(Uz*1e3, Uy*1e3, Ux*1e3, marker='.')
            ax.set_xlim([-50,50])
            ax.set_ylim([-50,50])
            ax.set_zlim([-50,50])
            ax.view_init(azim=0,elev=90)

            levels = np.linspace(minPotential,maxPotential,1000)
            cset = ax.contourf(X, Y, Z, levels, cmap=colormap)
            ax.clabel(cset, fontsize=9, inline=1)
            ax.set_xlabel("z (mm)",fontsize=16, labelpad=16)
            plt.xticks(rotation=45)
            plt.title("Acoustic Radiation Pressure", fontsize=16)
            plt.xticks(size=16)
            plt.show()
        else:
            print("Please define system geometry")

class masterframe(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.frames = {}
        self.drops = {}
        self.btns = {}
        self.drop_selections = {}

    def setouterproperties(self):
        self.geometry("750x850")
        self.iconbitmap(".\icon.ico")
        self.title("Simulation Platform for Acoustic Levitation Traps (SALT)")

    def placemainwidgets(self):
        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        pager = Pager()

        self.top_selection = tk.StringVar()
        self.top_selection.set("Select Type")
        self.frames[0] = tk.LabelFrame(self, text="Top component", padx=10, pady=10)
        self.frames[0].pack(padx=10, pady=10)
        self.frames[0].place(x=20, y=0)
```

```
self.drops[0] = tk.OptionMenu(self.frames[0], self.top_selection,
"Array", "Transducer",
command=lambda x:pager.toggle_page1(self.top_selection, self.frames[0]))
self.drops[0].pack()

self.bot_selection = tk.StringVar()
self.bot_selection.set("Select Type")
self.frames[1] = tk.LabelFrame(self, text="Bottom components", padx=10, pady=10)
self.frames[1].pack(padx=10, pady=10)
self.frames[1].place(x=420, y=0)
self.drops[1] = tk.OptionMenu(self.frames[1], self.bot_selection,
"Array", "Reflector", command=lambda x:pager.toggle_page2(
self.bot_selection, self.frames[1]))
self.drops[1].pack()

self.frames[2] = tk.LabelFrame(self, text="Medium Properties")
self.frames[2].pack(padx=5,pady=5,anchor=tk.CENTER)

sub_frame = tk.LabelFrame(self.frames[2], text="Speed of sound (m/s)")

self.medium_c = tk.Entry(sub_frame, width=8,justify=tk.CENTER)
self.medium_c.insert(0,343)
self.medium_c.pack(padx=5,pady=5,side=tk.BOTTOM)
sub_frame.pack(side=tk.LEFT)

sub_frame2 = tk.LabelFrame(self.frames[2], text="Zero-density (kg/m^3)")

self.medium_density = tk.Entry(sub_frame2, width=8, justify=tk.CENTER)
self.medium_density.insert(0,1.2)
self.medium_density.pack(padx=5,pady=5,side=tk.BOTTOM)
sub_frame2.pack(side=tk.RIGHT)

self.frames[3] = tk.LabelFrame(self, text="Options", padx=10, pady=10)
self.frames[3].pack(padx=10,pady=10,side=tk.BOTTOM)

self.btns[0] = tk.Button(self.frames[3], text="Display Geometry",
command=lambda:pager.render(self))
self.btns[0].pack(pady=10, padx =5, side=tk.LEFT)

self.btns[1] = tk.Button(self.frames[3], text="Run Simulation",
command=lambda:pager.compute_acoustic_rad(self))
self.btns[1].pack(pady=10, padx = 5, side=tk.LEFT)

self.colormap = tk.StringVar()
self.colormap.set("hot")
self.drops[3] = tk.OptionMenu(self.frames[3], self.colormap,
```

```
        "hot", "hsv", "jet", "winter", "bone")
        self.drops[3].pack(pady=10, padx = 5, side=tk.BOTTOM)
```

B The matrix method

B.1 matrices.py

This module contains methods which computes the different matrices involved in evaluating the matrix method

```
  import numpy as np

def DistanceMatrices(Vx, Vy, Vz, Ux, Uy, Uz, Mx, My, Mz):
    """Function computes the distance between measurement plane and components"""
    nT = len(Vx)
    nR = len(Ux)
    nM = len(Mx)

    Ax = np.repeat(Vx,nM,0).reshape(nT,nM).T
    Bx = np.repeat(Mx.reshape(nM,1),nT,axis=1)

    Ay = np.repeat(Vy,nM,0).reshape(nT,nM).T
    By = np.repeat(My.reshape(nM,1),nT,axis=1)

    Az = np.repeat(Vz,nM,0).reshape(nT,nM).T
    Bz = np.repeat(Mz.reshape(nM,1),nT,axis=1)

    r_nm = np.sqrt((Bx-Ax)**2 + (By-Ay)**2 + (Bz-Az)**2)

    Ax = np.repeat(Ux,nM,0).reshape(nR,nM).T
    Bx = np.repeat(Mx.reshape(nM,1),nR,axis=1)

    Ay = np.repeat(Uy,nM,0).reshape(nR,nM).T
    By = np.repeat(My.reshape(nM,1),nR,axis=1)

    Az = np.repeat(Uz,nM,0).reshape(nR,nM).T
    Bz = np.repeat(Mz.reshape(nM,1),nR,axis=1)

    r_im = np.sqrt((Bx-Ax)**2 + (By-Ay)**2 + (Bz-Az)**2)

    Ax = np.repeat(Ux, nT, 1)
    Bx = np.repeat(Vx.T, nR, 0)

    Ay = np.repeat(Uy, nT, 1)
    By = np.repeat(Vy.T, nR, 0)
```

```
    Az = np.repeat(Uz, nT, 1)
    Bz = np.repeat(Vz.T, nR, 0)

    r_in = np.sqrt((Bx-Ax)**2 + (By-Ay)**2 + (Bz-Az)**2)
    r_ni = r_in.T

    return r_nm, r_im, r_in, r_ni

def TransferMatrices(
mediumProperties, zPosProperties, zNegProperties, r_nm, r_im, r_in, r_ni):
    """ Method for computing transfer matrices """
    Sn = 1e-6
    Si = 1e-6

    c = mediumProperties["SpeedOfSound"]
    f1 = zPosProperties["TransFreq"]

    if "TransFreq" in zNegProperties:
        f2 = zNegProperties["TransFreq"]
    else:
        f2 = 0

    if f1 != 0:
        wL1 = c/f1
        kk1 = 2*np.pi/wL1
    elif f1 == 0:
        wL1 = 0
        kk1 = 0
    if f2 != 0:
        wL2 = c/f2
        kk2 = 2*np.pi/wL2
    elif f2 == 0:
        wL2 = 0
        kk2 = 0

    if zPosProperties["Type"] == "Array" and zNegProperties["Type"] == "Array":
        T_TR = Sn*np.exp(-1j*kk1*r_ni - 1j*kk2*r_in)/(r_in)
        T_RT = Si*np.exp(-1j*kk1*r_in - 1j*kk2*r_ni)/(r_ni)
        T_RM = Sn*np.exp(-1j*kk1*r_im)/r_im
        T_TM = Si*np.exp(-1j*kk2*r_nm)/r_nm
    elif zPosProperties["Type"] == "Array" and zNegProperties["Type"] =="Reflector":
        T_TR = Sn*np.exp(-1j*kk1*r_in)/(r_in)
        T_RT = Si*np.exp(-1j*kk1*r_ni)/(r_ni)
        T_RM = Sn*np.exp(-1j*kk1*r_im)/(r_im)
        T_TM = Si*np.exp(-1j*kk1*r_nm)/(r_nm)
```

```
    elif zPosProperties["Type"] == "Transducer" and zNegProperties["Type"] =="Reflector":
        T_TR = Sn*np.exp(-1j*kk1*r_in)/(r_in)
        T_RT = Si*np.exp(-1j*kk1*r_ni)/(r_ni)
        T_RM = Sn*np.exp(-1j*kk1*r_im)/(r_im)
        T_TM = Si*np.exp(-1j*kk1*r_nm)/(r_nm)

    return T_TR, T_RT, T_RM, T_TM
```

B.2 geometries.py

This module contains methods which generates the different components of an acoustic levitation device.

```
  import numpy as np

def CreateArray(properties):
    """ Method for constructing an array of transducers """
    transducer_radius = 4.5*1e-3
    transducer_per_layer = properties["Depth"]
    layers = properties["Layers"]
    socket_radius = properties["Radius"]
    r_c = properties["RadiusCurvature"]
    h = properties["Displacement"]
    s = properties["Orientation"]

    tz0 = s*(h - r_c)

    X, Y = np.mgrid[-socket_radius*1e3:socket_radius*1e3,
    -socket_radius*1e3:socket_radius*1e3]
    X = X*1e-3
    Y = Y*1e-3
    r = np.array([24, 46, 68, 90, 112, 135, 158, 180])*1e-3
    Vx = []
    Vy = []

    for kk in range(layers):
        n = int(transducer_per_layer[kk])
        beta = np.linspace(2*np.pi/n, 2*np.pi, n)

        for ii in range(n):
            C = np.sqrt((X - 0.5*r[kk]*np.cos(beta[ii]))**2 +
            (Y - 0.5*r[kk]*np.sin(beta[ii]))**2) <= transducer_radius
            vec1 = X[C]
            vec2 = Y[C]
            Vx = np.append(Vx, vec1)
```

```
            Vy = np.append(Vy, vec2)

        if properties["Concave"]:
            Vz = tz0 + s*np.sqrt(r_c**2 - np.power(Vx,2) - np.power(Vy,2))
        elif not properties["Concave"]:
            Vz = s*h*np.ones([len(Vx),1])

    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])
    S = Vx + Vz + Vy

    if np.isnan(S).any():
        Vx = np.delete(Vx,np.argwhere(np.isnan(S)))
        Vy = np.delete(Vy,np.argwhere(np.isnan(S)))
        Vz = np.delete(Vz,np.argwhere(np.isnan(S)))

    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])

    return Vx, Vy, Vz

def CreateMedium(Vx,Vz,Ux,Uz):
    """Function generates points in a plane between
    components where pressure will be calculated"""
    xMax = np.max(Ux)+5e-4
    xMin = -np.max(Ux)-5e-4
    zMax = np.min(Vz)-5e-4
    zMin = np.max(Uz)+5e-4

    x_span = np.arange(xMin, xMax, 1e-3)
    z_span = np.arange(zMin, zMax, 1e-3)

    Mx, Mz = np.meshgrid(x_span, z_span, sparse=False)

    sz = Mx.shape
    My = np.zeros([sz[0],sz[1]])

    szx = Mx.shape
    Mx = Mx.reshape(szx[0]*szx[1],1)

    szy = My.shape
    My = My.reshape(szy[0]*szy[1],1)

    szz = Mz.shape
```

```
    Mz = Mz.reshape(szz[0]*szz[1],1)

    return Mx, My, Mz, x_span, z_span

def CreateReflector(properties):
    """ Method for constructing a reflector """
    R = properties["Radius"]*1e3
    r_c = properties["RadiusCurvature"]
    s = properties["Orientation"]
    d = properties["Displacement"]
    z0 = s*d

    R_span = np.arange(-R*1e-3,R*1e-3,1e-3)
    R_length = len(R_span)

    X, Y = np.mgrid[-R:R, -R:R]
    X = X*1e-3
    Y = Y*1e-3
    Z = np.zeros([R_length, R_length])

    rows, cols = np.mgrid[0:R_length, 0:R_length]
    C = np.sqrt((rows-R)**2 + (cols-R)**2)<R

    Vx = X[C]
    Vy = Y[C]
    if properties["Concave"]:
        if s == -1:
            Vz = z0 - np.sqrt(r_c**2 - (Vx)**2 - (Vy)**2) + r_c
        elif s == 1:
            Vz = z0 + np.sqrt(r_c**2 - (Vx)**2 - (Vy)**2) - r_c
        else:
            Vz = 0
    elif not properties["Concave"]:
        if s == -1:
            Vz = z0*np.ones([len(Vx),1])
        elif s == 1:
            Vz = z0*np.ones([len(Vx),1])
        else:
            Vz = 0


    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])
    S = Vx + Vz + Vy
```

```
    if np.isnan(S).any():
        Vx = np.delete(Vx,np.argwhere(np.isnan(S)))
        Vy = np.delete(Vy,np.argwhere(np.isnan(S)))
        Vz = np.delete(Vz,np.argwhere(np.isnan(S)))

    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])

    return Vx, Vy, Vz

def CreateTransducer(properties):
    """ Method for constructing a single transducer """
    R = properties["Radius"]*1e3
    r_c = properties["RadiusCurvature"]
    s = properties["Orientation"]
    d = properties["Displacement"]

    z0 = s*d

    R_span = np.arange(-R*1e-3,R*1e-3,1e-3)
    R_length = len(R_span)

    X, Y = np.mgrid[-R:R,-R:R]
    X = X*1e-3
    Y = Y*1e-3
    Z = np.zeros([R_length, R_length])

    rows, cols = np.mgrid[0:R_length, 0:R_length]
    C = np.sqrt((rows-R)**2 + (cols-R)**2)<R

    Vx = X[C]
    Vy = Y[C]

    if properties["Concave"]:
        if s == -1:
            Vz = z0 - np.sqrt(r_c**2 - (Vx)**2 - (Vy)**2) + r_c
        elif s == 1:
            Vz = z0 + np.sqrt(r_c**2 - (Vx)**2 - (Vy)**2) - r_c
        else:
            Vz = 0
    elif not properties["Concave"]:
        if s == -1:
            Vz = z0*np.ones([len(Vx),1])
        elif s == 1:
            Vz = z0*np.ones([len(Vx),1])
```

```
        else:
            Vz = 0

    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])
    S = Vx + Vz + Vy

    if np.isnan(S).any():
        Vx = np.delete(Vx,np.argwhere(np.isnan(S)))
        Vy = np.delete(Vy,np.argwhere(np.isnan(S)))
        Vz = np.delete(Vz,np.argwhere(np.isnan(S)))

    Vx = Vx.reshape([len(Vx),1])
    Vy = Vy.reshape([len(Vy),1])
    Vz = Vz.reshape([len(Vz),1])

    return Vx, Vy, Vz
```

B.3 computation.py

This module contains methods which carry out the algebra of the matrix method.

```
  import numpy as np

def ComputePressure(mediumProperties,T_TR,T_RT,T_RM,T_TM,
zPosProperties,zNegProperties,nT,nR,nM):
    """ Method computes pressure matrix in accordance with the matrix method """

    t1 = zPosProperties["Phase"]
    f1 = zPosProperties["TransFreq"]

    if "Phase" in zNegProperties:
        t2 = zNegProperties["Phase"]
    else:
        t2 = 0

    if "TransFreq" in zNegProperties:
        f2 = zNegProperties["TransFreq"]
    else:
        f2 = 0

    d1 = 1e-6
    d2 = 1e-6

    rho = mediumProperties["Density"]
```

```
c = mediumProperties["SpeedOfSound"]
t = 0

if f1 != 0:
    wL1 = c/f1
    omega1 = 2 * np.pi * f1
    U1 = np.ones([nT, 1])*d1*np.exp(-1j*(omega1*t + np.pi*t1))
    A1 = (1j/wL1)
    C1 = omega1*rho*c/wL1
if f1 == 0:
    wL1 = 0
    omega1 = 0
    U1 = np.zeros([nT, 1])
    A1 = 0
    C1 = 0

if f2 != 0:
    wL2 = c/f2
    omega2 = 2 * np.pi * f2
    U2 = np.ones([nR, 1])*d2*np.exp(-1j*(omega2*t + np.pi*t2))
    A2 = (-1j/wL2)
    C2 = omega2*rho*c/wL2
if f2 == 0:
    wL2 = 0
    omega2 = 0
    U2 = np.zeros([nR, 1])
    A2 = 0
    C2 = 0

if zPosProperties["Type"] == "Transducer" and zNegProperties["Type"] == "Reflector":
    PT0 = (C1)*T_TM@U1;
    PT1 = (C1)*(A1)*T_RM@T_TR@U1;
    PT2 = (C1)*(A1**2)*T_TM@T_RT@T_TR@U1;
    PT3 = (C1)*(A1**3)*T_RM@T_TR@T_RT@T_TR@U1;
    PT4 = (C1)*(A1**4)*T_TM@T_RT@T_TR@T_RT@T_TR@U1;
    PT5 = (C1)*(A1**5)*T_RM@T_TR@T_RT@T_TR@T_RT@T_TR@U1;
    PT6 = (C1)*(A1**6)*T_TM@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@U1;
    PT7 = (C1)*(A1**7)*T_RM@T_TR@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@U1;
    PT8 = (C1)*(A1**8)*T_TM@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@U1;
    PT9 = (C1)*(A1**9)*T_RM@T_TR@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@T_RT@T_TR@U1;
    PT = PT0 + PT2 + PT4 + PT6 + PT8
    PR = PT1 + PT3 + PT5 + PT7 + PT9
elif (zPosProperties["Type"] == "Array" and zNegProperties["Type"] == "Array") or
(zPosProperties["Type"] == "Transducer" and zNegProperties["Type"] == "Array"):
    PT = (C1)*T_TM@U1
    PR = (C2)*T_RM@U2
```

```
    elif zPosProperties["Type"] == "Array" and zNegProperties["Type"] == "Reflector":
        PT = (C1)*T_TM@U1
        PR = (C1)*(A1)*T_RM@T_TR@U1
    P = PT + PR
    return P

def ComputeRelativePotential(Ptotal, mediumProperties, zPosProperties, zNegProperties):
    """ Method computes the relative acoustic potential """
    c = mediumProperties["SpeedOfSound"]
    rho = mediumProperties["Density"]
    f = zPosProperties["TransFreq"]
    w = c/f
    p = np.real(Ptotal)
    sz = Ptotal.shape
    M = sz[0]*sz[1]
    phi = -Ptotal/(1j*w*rho)

    gradx, grady = np.gradient(phi,5e-4,5e-4)
    gradP = np.sqrt(gradx**2 + grady**2)
    T1 = (np.real(Ptotal*np.conj(Ptotal))/M)/(3*rho*c**2)
    T2 = -0.5*rho*(np.real(np.multiply(gradP,np.conj(gradP)))/M)
    potential = T1+T2

    return potential
```

B.4 MatrixMethod.py

This module contains methods which call the methods in the modules presented in the previous subsections in the right order to evaluate the matrix method for the parameters defined by the input from the GUI.

```
  import time, sys
import numpy as np
from geometries import CreateTransducer, CreateReflector, CreateArray, CreateMedium
from matrices import DistanceMatrices, TransferMatrices
from computation import ComputePressure, ComputeRelativePotential

def MatrixMethod(mediumProperties,zPosProperties,zNegProperties):

    if zPosProperties["Type"] == "Array":
        print("Creating array...")
        start = time.time()
        Vx, Vy, Vz = CreateArray(zPosProperties)
        end = time.time()
        diff1 = end - start
        print("Create array took %.6f" % diff1, "seconds")
```

```
elif zPosProperties["Type"] == "Transducer":
    print("Creating Transducer...")
    start = time.time()
    Vx, Vy, Vz = CreateTransducer(zPosProperties)
    end = time.time()
    diff1 = end - start
    print("Create transducer took %.6f" % diff1, "seconds")
else:
    Vx = 0
    Vy = 0
    Vz = 0

if zNegProperties["Type"] == "Array":
    print("Creating array...")
    start = time.time()
    Ux, Uy, Uz = CreateArray(zNegProperties)
    end = time.time()
    diff2 = end - start
    print("Create array took %.6f" % diff2, "seconds")

elif zNegProperties["Type"] == "Reflector":
    print("Creating reflector...")
    start = time.time()
    Ux, Uy, Uz = CreateReflector(zNegProperties)
    end = time.time()
    diff2 = end - start
    print("Create reflector took %.6f" % diff2, "seconds")
else:
    Ux = 0
    Uy = 0
    Uz = 0

print("Creating medium...")
start = time.time()
Mx, My, Mz, x_span, z_span = CreateMedium(Vx, Vz, Ux, Uz)
end = time.time()
diff3 = end - start
print("Creating medium took %.6f" % diff3, "seconds")

print("Computing distance matrices...")
start = time.time()
r_nm, r_im, r_in, r_ni = DistanceMatrices(Vx, Vy, Vz, Ux, Uy, Uz, Mx, My, Mz)
end = time.time()
diff4 = end - start
print("Computing distance matrices took %.6f" % diff4, "seconds")
```

```
    print("Computing transfer matrices...")
    start = time.time()
    T_TR, T_RT, T_RM, T_TM = TransferMatrices(mediumProperties, zPosProperties, zNegProperti
                             r_nm, r_im, r_in, r_ni)
    end = time.time()
    diff5 = end - start
    print("Computing transfer matrices took %.6f" % diff5, "seconds")

    nT = len(Vx)
    nR = len(Ux)
    nM = len(Mx)
    print("Computing pressure matrix...")
    start = time.time()
    pressure = ComputePressure(mediumProperties,T_TR,T_RT,T_RM,T_TM,
                zPosProperties,zNegProperties,nT,nR,nM)
    end = time.time()
    diff6 = end - start
    print("Computing pressure matrices took %.6f" % diff6, "seconds")

    x = len(x_span)
    z = len(z_span)
    P_ = pressure.reshape([z, x])

    print("Computing relative acoustic potential...")
    start = time.time()
    relative_potential = ComputeRelativePotential(P_, mediumProperties, zPosProperties,
                             zNegProperties)
    end = time.time()
    diff7 = end - start
    print("Computing relative acoustic potential took %.6f" % diff7, "seconds")

    c = mediumProperties["SpeedOfSound"]
    rho = mediumProperties["Density"]

    # acoustic_radiation_pressure = (np.real(pressure)**2/(4*rho*c**2)).reshape([z, x])
    acoustic_radiation_pressure = (np.real(P_)**2)/(4*rho*c**2)
    return acoustic_radiation_pressure, relative_potential, np.real(P_), x_span, z_span

def CreateGeometry(zPosProperties, zNegProperties):
    if zPosProperties["Type"] == "Array":
        Vx, Vy, Vz = CreateArray(zPosProperties)
    elif zPosProperties["Type"] == "Transducer":
        Vx, Vy, Vz = CreateTransducer(zPosProperties)
    else:
        Vx = 0
```

```
    Vy = 0
    Vz = 0

if zNegProperties["Type"] == "Array":
    Ux, Uy, Uz = CreateArray(zNegProperties)
elif zNegProperties["Type"] == "Reflector":
    Ux, Uy, Uz = CreateReflector(zNegProperties)
else:
    Ux = 0
    Uy = 0
    Uz = 0
return Ux, Uy, Uz, Vx, Vy, Vz
```