ENGPHYS 2CM4 Design Project Report

17 April 2023

Daniel Kogan (400364625)

Ojas Chaudhary (400370047)

Mark Jonathan (400392319)

Shyavan Sridhar (400388748)

Mohammed Aasim (400294890)

Contents

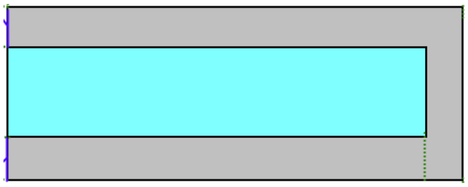
Problem Definition

The problem investigated for this project is based off the shell that was provided in [1]. A combined piezoelectric-thermal resonant cantilever is designed to achieve as high a transverse tip displacement as possible. The cantilever will consist of a piezoelectric material with a layer of metal wrapped around the tip and on the top and bottom faces. The metal deposited will be varied to identify any differences observed through multiple material properties. These metals are titanium, gold, brass, and the fictional metal "vibranium". Titanium and gold have both been worked with in this course before and as such, can be used as indicators of the model's performance in a control manner. However, brass was chosen for its resonance properties as it is often used in musical instruments and vibranium for its fictional ability to absorb nearly all vibrations and kinetic energy directed at it as an interesting experiment. As vibranium is a fictional material, it will be approximated by a Tantalum alloy with a high Young's Modulus and Poisson's Ratio and a very low density as it is stated in the Marvel films to be one-third the weight of steel for the same volume. The thickness of the metal layer on the top, bottom, and side were all independently varied to identify the largest tip displacement and the equilibrium temperature at the junction between the tip of the piezoelectric and metal is to be taken and reported. The resonant frequency of such a cantilever is also to be found for the best performing dimensions for each material. The cantilever will be actuated by a fixed voltage across the metal contacts at the anchor producing movement through the piezoelectric effect and internal heat generation in the metal layer. Moreover, there will also be convection from the surrounding air at a set temperature and the anchor will be made of an insulating material. A diagram from [2] shows the piezoelectric wrapped in a metal layer.

The specific properties for the piezoelectric and vibranium components are as follows. The remaining materials' properties were taken as averages from standard materials databases [3, 4]:

1. Piezoelectric (properties taken from DA4 [5])
   a. Dimensions are 200 μm by 500 μm by 2 mm
   b. The top electrical contact is at 1 V relative to the bottom contact
   c. It is anisotropic with a compliance matrix of

$$S = \begin{bmatrix} 13 & -2 & -1.2 & -5 & 0 & 0 \\ -2 & 13 & -1.2 & 5 & 0 & 0 \\ -1.2 & -1.2 & 10 & 0 & 0 & 0 \\ -5 & -5 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & -9 \\ 0 & 0 & 0 & 0 & -9 & 30 \end{bmatrix} pm/m \cdot Pa$$



   d. It has a piezoelectric strain coupling matrix of $d = \begin{bmatrix} 20 & -24 & 0 & 0 & 0 & 0 \\ 16 & -35 & 0 & 0 & 0 & 0 \\ 45 & 0 & 70 & 0 & 0 & 0 \end{bmatrix} pm/V$
   e. It has a permittivity matrix of $\varepsilon_T = \varepsilon_0 \begin{bmatrix} 4.52 & 0 & 0 \\ 0 & 4.52 & 0 \\ 0 & 0 & 4.68 \end{bmatrix} F/m$ where $\varepsilon_0 = 8.85 \times 10^{-12}\ F/m$
   f. Thermal Expansion Coefficient of $\alpha_T = 1.15\ \mu m/K$ and density of $\rho_M = 2.65\ g/cm^3$
2. The ambient temperature surrounding the structure is 25°C and it also begins at this temperature
3. Vibranium (taken from [6])
   a. Isotropic with $E = 300\ GPa, \nu = 0.49, \alpha_T = 9.5 \times 10^{-6}\ K^{-1}, \sigma_E = 1.3 \times 10^{-7} \frac{S}{m}, \rho_M = 0.497 \frac{g}{cm^3}$ (all sides of metal should also be between 50 μm and 500 μm thick)

Analysis and Testing

Background Theory and Development of The Analytical Model

The goal of this project is to determine the optimal thicknesses of the metal portions wrapping around the piezoelectric to maximize the transverse tip displacement under the above constraints. The techniques learned throughout this course and present in [7-10] can all be employed to solve this problem. Moreover, a very similar problem is presented in [2] and can be used as a base for the model as well as the previous models developed for previous design assignments. In particular, Hooke's Law for stresses and the relationships that determine strains due to electric fields and temperature changes will need to be utilized. Moreover, the steady-state heat equation knowledge from [8] and the modal analysis knowledge form [10] are also required to find the junction temperature and resonant frequency at the first mode respectively. The code from DA4 in particular can be useful as it involves a similar cantilever and could be used for validation purposes as shown later.

As the objective is to obtain transverse tip displacement data of the structure, this problem can largely be simplified into 2D without any negative consequences or lack of information. The transverse tip displacement does not change across the y axis and so, this dimension can be eliminated, leaving only the x and z axes requiring simulation, greatly decreasing the computational and temporal costs required to solve the problem. To accomplish this, any elements pertaining to the y axis, or width, can be removed from the provided material properties and both Hooke's Law and the strain relations can be simplified to two dimensions. This will require any properties pertaining to the axial x and z directions as well as the shear x-z direction. Thus, the elements from each matrix that should be used are those in rows and columns 1, 3, and 5. The reasons for removing the y axis and not the x or z axis will be covered shortly.

To ensure consistency and prevent unit-conversion errors, all units were first converted to SI for use within the model. The displacement is also produced by the model in m as a result but is converted to µm when output to the user to increase readability.

In order to solve this problem, the first step would be to obtain the stiffness matrices for both materials to be used with Hooke's Law. For the piezoelectric, this can be done by taking the inverse of the provided compliance matrix as $C = S^{-1}$ where C is the stiffness matrix. In the case of the polymer, as the material is isotropic, its provided Young's Modulus and Poisson's Ratio can be used to find the stiffness matrix as follows.

$$C = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} (1-\nu) & \nu & \nu & 0 & 0 & 0 \\ \nu & (1-\nu) & \nu & 0 & 0 & 0 \\ \nu & \nu & (1-\nu) & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{(1-2\nu)}{2} \end{bmatrix}$$

From here, Hooke's Law can be used to relate the strains to the stresses as follows.

$\vec{\sigma} = C\vec{\varepsilon}_m$
$\vec{\varepsilon}_m = \vec{\varepsilon} - \ \alpha_T\Delta_T - d\vec{E}$

Where $\vec{\sigma}$ is the stress vector, $\vec{\varepsilon}_m$ is the mechanical strain, $\vec{\varepsilon}$ is the strain as defined directly as the change in displacement in each dimension for both axial and shear, Δ_T is the temperature difference as presented in the problem, and $\vec{E}$ is the electric field vector defined by the voltage applied over the distance in that dimension. To incorporate the resonant frequency, a separate script can be made in a similar manner to DA5 in order to perform modal analysis and simplify the simulation process.

As the piezoelectric and relations are the same from DA4, the code from DA4 can be started with and modified to allow for a region set up before the piezoelectric such that it can encompass the entirety of the metal, with its center portion overridden by the piezoelectric region coming after. From here, the techniques from [2] can be incorporated to account for the forward piezoelectric effect and heat transfer. However, instead of fixing the anchor at a temperature, the anchor will be set to be insulated and a load constraint on the temperature will also be applied on the remaining sides to represent convection as in DA2 [8]. The electric field must also be modified to pass across the contacts at the anchor such that the electric field flows through the metal portion in a curve around the tip as in [2].

Knowing all of this, the model can now be constructed based on previously existing code with an excerpt referring to the important theory previously explained presented below. For the full code please refer to the appendices at the end of the document.

```
! Equations
E_field = -grad(V)                  ! Electric Field (V/m)
J = sigma_e*E_field                 ! Current Density (A/m^2)
q_dot = -k*grad(Temp)               ! Heat Flux for Conduction (W/m^2)
q_dotc = h_air*(T_inf - Temp)       ! Heat Flux for Convection (W/m^2)
q_dotvol = dot(J, E_field)          ! Volumetric Heat Generation (W/m^3)

E_x = xcomp(E_field)                ! Electric Field in x (V/m)
E_z = zcomp(E_field)                ! Electric Field in z (V/m)

! Strain Definitions from Displacements (unitless ratio)
epsilon_x = dx(u)
epsilon_z = dz(w)
gamma_xz = dx(w) + dz(u)

! Mechanical Strain (unitless ratio)
epsilon_xm = epsilon_x - alpha_T*Delta_T - (d_11*E_x + d_31*E_z)
epsilon_zm = epsilon_z - alpha_T*Delta_T - (d_13*E_x + d_33*E_z)
gamma_xzm = gamma_xz - (d_15*E_x + d_35*E_z)

! Hookes Law for Stresses (Pa)
s_x = C_11*epsilon_xm + C_13*epsilon_zm + C_15*gamma_xzm
s_z = C_31*epsilon_xm + C_33*epsilon_zm + C_35*gamma_xzm
s_xz = C_51*epsilon_xm + C_53*epsilon_zm + C_55*gamma_xzm

! Electric Displacement Field due to Piezoelectric Effect (C/m^2)
D_piezox = d_11*s_x + d_13*s_z + d_15*s_xz
D_piezoz = d_31*s_x + d_33*s_z + d_35*s_xz
```

```
D_piezo = vector(D_piezox, 0, D_piezoz)

! Total Electric Displacement Field (C/m^2)
D_field = epsilon_0*epsilon_r*E_field + D_piezo

EQUATIONS                           ! PDEs; one for each variable
V: div(J) = 0
rho_free: div(D_field) = rho_free
u: dx(s_x) + dz(s_xz) = 0
w: dx(s_xz) + dz(s_z) = 0
Temp: q_dotvol = div(q_dot)

BOUNDARIES
      REGION "Metal"
                  alpha_T = 8.6e-6  ! Coefficient of Thermal Expansion (K^-1)
                  k = 17            ! Thermal Conductivity (W/m K^-1)

                  epsilon_r = 1e6   ! Relative Permittivity (unitless ratio)
                  sigma_e = 1/4.2e-7      ! Electrical Conductivity (S/m)

                  ! Piezoelectric Coupling Coefficients (m/V)
                  d_11 = 0 d_13 = 0 d_15 = 0
                  d_31 = 0 d_33 = 0 d_35 = 0

                  ! Stiffness Matrix (Pa^-1)
                  C_11 = E_metal*(1 - nu_metal)/((1 + nu_metal)*(1 -
2*nu_metal)) C_13 =  E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal)) C_15
= 0
                  C_31 = E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal))
                              C_33 = C_11       C_35 = 0
                  C_51 = 0    C_53 = 0    C_55 = E_metal/(2*(1 + nu_metal))

                  START (0, 0)
                        load(Temp) = q_dotc     ! Set up convection on all
                                                free sides
                  LINE TO (l_x + t_metal_side, 0)
                  LINE TO (l_x + t_metal_side, t_piezo + t_metal_top +
t_metal_bottom)
                  LINE TO (0, t_piezo + t_metal_top + t_metal_bottom)
                                    ! Hold one side fixed for cantilever
                        value(V) = V_max  ! Set V to V_max on top contact
                        value(rho_free) = 0
                                  ! Set free charge density to zero on contact
                        load(Temp) = 0    ! Thermally insulate fixed side
                        value(u) = 0
                        value(w) = 0
                  LINE TO (0, t_piezo/2 + t_metal_bottom)
                              ! Hold one side fixed for cantilever
                        value(V) = 0      ! Set V to 0 on bottom contact
```

Validation

The model developed can be validated using a few different methods. First, the piezoelectric portion of by setting up the model to use the same conditions and material properties as those of DA4 and identifying the displacement, the geometry of the model can be validated. This was done as an initial check to ensure the model was on the right track by taking the already validated DA4 code, modifying the polymer region to match the metal region setup shown above albeit with the polymer's properties and positioning and finding the displacement produced by the piezoelectric and thermal expansion alone. This should be in line with the original DA4 code, and it was, as both were around 3.54e-7 m in displacement for the test case used in that project. Knowing that all of this was valid, the more complex portions could be validated.

The next portion of the model to validate can be done via the plotting functionality. The electric field and voltage plots can show if the field is being applied properly and if the cantilever is responding correctly.

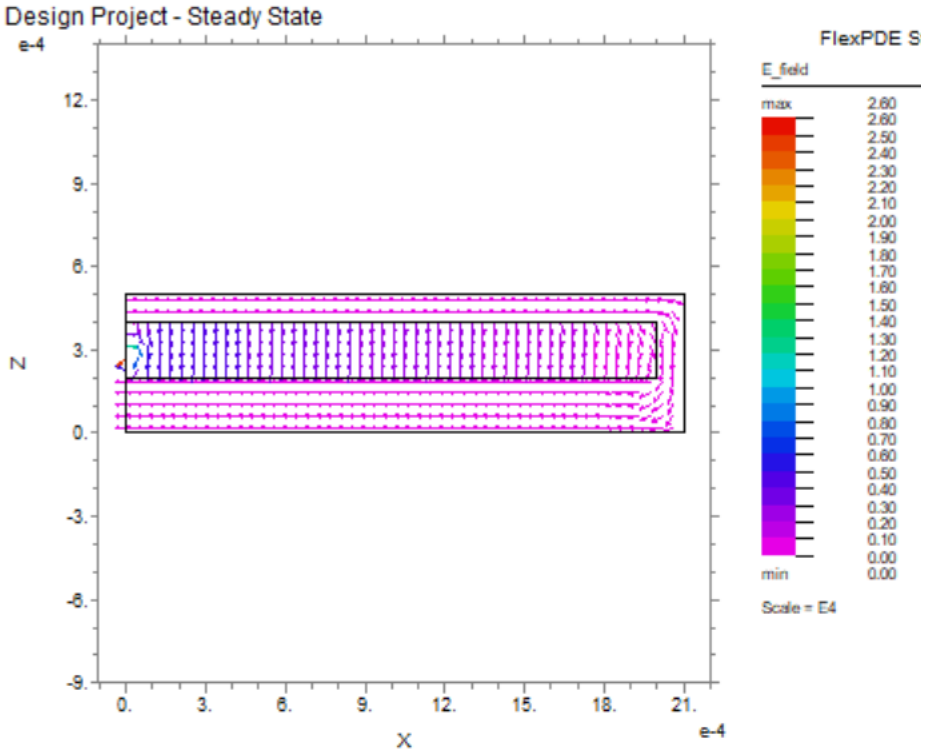


Figure 1: Electric Field Vector Plot

The plot of the electric field above shows how there is little to no field in the conductor which is as expected. Most of the field goes through the piezoelectric which is acting as a dielectric. However, the direction of the field also indicates that the field is being applied in the correct direction. There is field going across the piezoelectric from top to bottom in addition to the zero vectors pointing across the conductor from the top left contact to the bottom left contact and through the tip which is the desired flow.

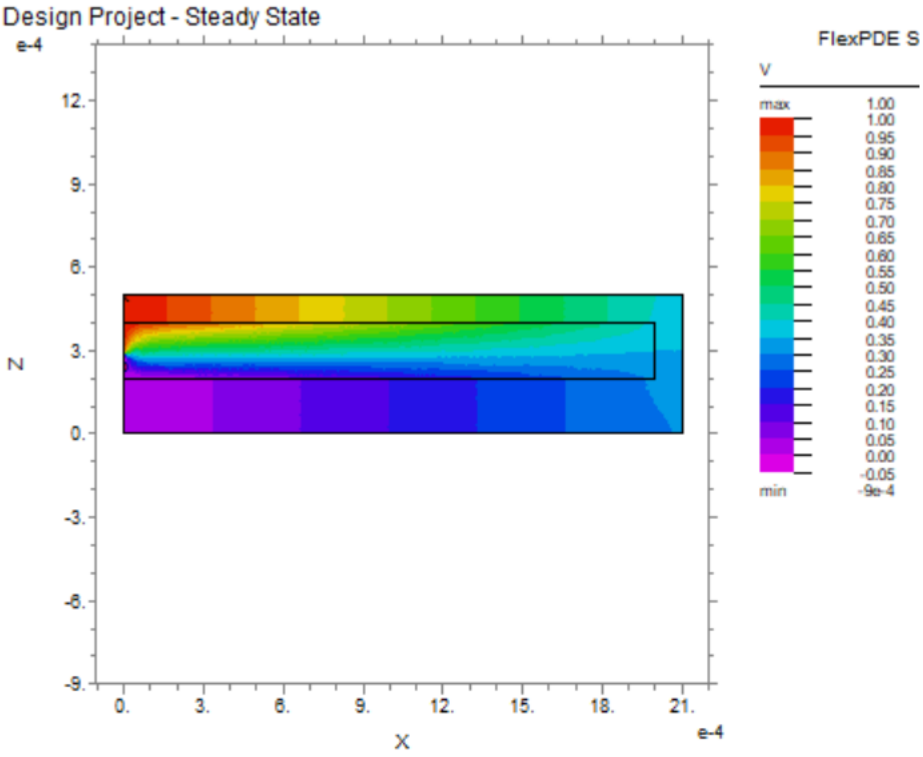


Figure 2: Voltage Contour Plot

The voltage graph above also shows further proof of the model working as closer to the left where the dielectric is between the metal layers, the structure acts as a capacitor with the dielectric having an even spread from the two voltages across the gradient. However, this spread becomes distorted further down the right as the contacts are on the left and the voltage continues to drop across the conductor. At the point where the piezoelectric is not present, the gradient is all but nearly gone as the metal nearby is at an equipotential.

The thermal stresses can also be validated by plotting the heat flux and temperature. As evidenced by the plot below, the heat flux is generally across the conductor and outwards from the structure. This is expected as the cooler air is taking some of the heat energy generated by the cantilever and the conductor is much better at carrying the heat, hence the most flux is in the conductor and at the edges. Moreover, there is not much escaping near the anchor which is because it was insulated.

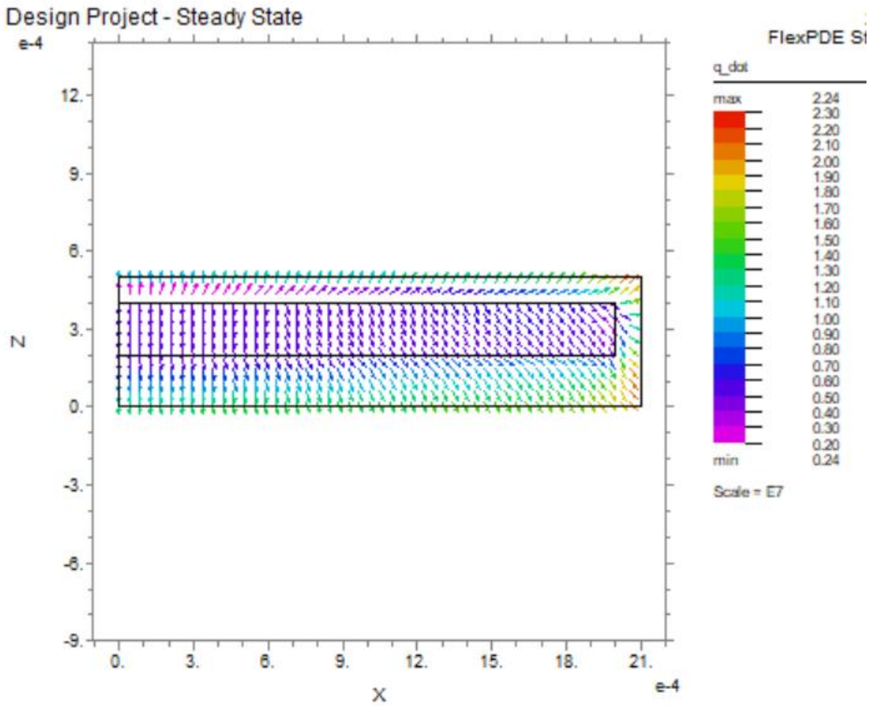


Figure 3: Heat Flux Vector Plot

The temperature plot also shows how the temperature is highest near the anchor and top corner where the conductor is thinner and so is able to dissipate less heat. The lowest temperature is found at the tip and in the thicker part of the conductor.

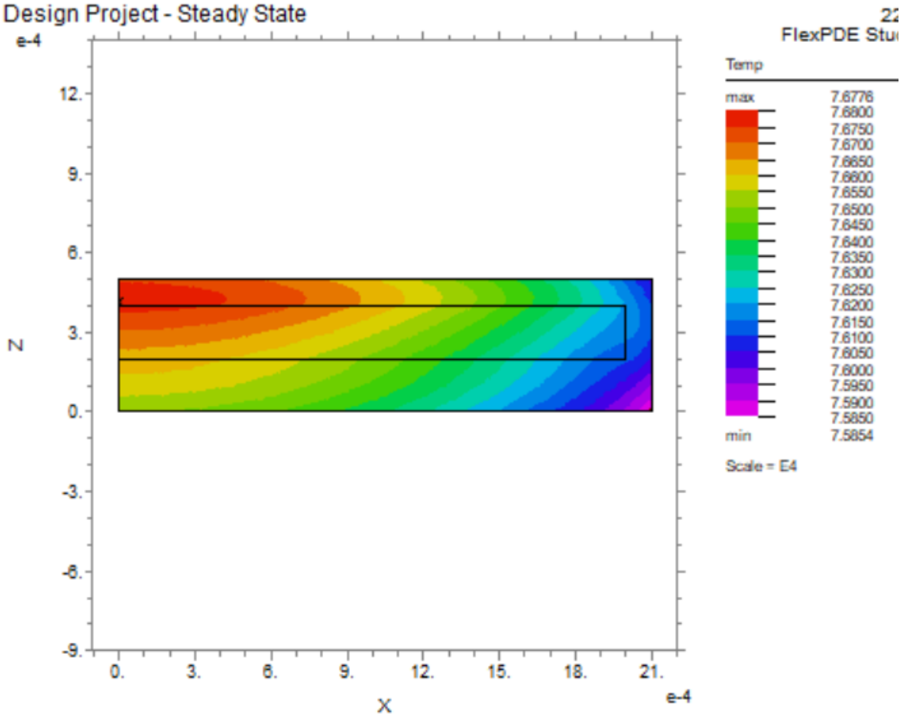


Figure 4: Temperature Contour Plot

The modal analysis model, having been mostly simple modifications to the material properties to the DA5 code is for the most part already validated. The change in geometry is taken to be validated from the above and the remaining is quite straight forward so as to not require too much explanation. Finding the first mode frequency can be done using the methods shown in [11]. Moreover, due to the nature of modal analysis, the resonant frequency is dependent on the shape and material of the structure and not any external stresses applied. As such, while thermal expansion and piezoelectric effect are included to drive the displacement of the beam, the magnitude of the electric field and the temperature difference experienced are inconsequential to the resonant frequency. This can be confirmed by changing the values to any arbitrary value at all and noting that the frequency output does not change. Thus, from the above evidence it can be taken that the model is at minimum accurate to a certain degree.

Scripting and Optimization

Scripting

As the FlexPDE model has been validated, the Python code can be constructed to sweep through acceptable thicknesses for the polymer layer when depositing it on each side. However, the FlexPDE code must be modified to accept various parameters from Python and output the transverse tip displacement, steady-state temperature, and resonant frequency in text files. Several lines are adjusted to contain placeholders in order to do this such that they can easily be adjusted with values supplied by Python. This is particularly used for changing material properties for the metal layer aside from the main use of sweeping through the thickness values.

The FlexPDE code modified can now be incorporated into the Python code, which is compartmentalized into multiple modules to allow easy repurposing of the code for future design assignments. The FlexPDE code, as well as any other global variables and constants, are placed in constants.py, any functions pertaining to running FlexPDE and sweeping through it are contained in auto_sweep.py, any plotting-related functions are contained in advanced_plotter.py, and the main script is located in solver.py. The code is mostly the same from the previous design assignments, with minor changes to variable names and the most changes to the constants in constants.py. For the entire code, refer to the appendices at the end of this document.

While time dependance as in [8] was explored in an attempt to identify how the cantilever changed in temperature and deformation over time, this was scrapped due to the simulation even in the time-independent form taking very long to run and output answers. The time to run also prevented further optimization of the model further explained in the optimization section. This time was slightly reduced by implementing the scripts in a multithreaded fashion but did not prove to provide a major boost even on an octa-core CPU.

In order to generate the range of component widths to sweep through, the constraints of a maximum and minimum polymer thickness of 2 mm and 20 µm respectively were taken into account. Using the linspace function within the numpy module, a set of 20 datapoints were created in each run of the program between this entire range initially, with progressively smaller ranges used on subsequent runs to further increase the resolution near the maximum and better fit a curve to the data present. The ngrid could also be made higher to yield more accurate results however, for the purposes of simulation, the ngrid was kept to 13 unless certain scripts explicitly failed and produced unusable data. This was the case for vibranium and as such the ngrid was increased to 20. However, due to time constraints, the other simulations were not repeated and so it should be noted that accuracy can be made significantly better with high ngrid values which will be clear in the results.

From here, the code can then be made to sweep through every thickness limited to 7 points in each dimension to identify the maximum displacement. This produced a total range of 343 points per material in a 3D scatter plot with a colour map associated with the displacement. Thus, to work within the limits of the software's student edition, and avoid reducing the ngrid greatly, the values fed to flex and produced by linspace from numpy were rounded to the precision of 1 micron. This produced better results and was further reinforced by manufacturing tolerances that would be present, preventing the fabrication of such a sensor at the much greater level of precision Python was generating datapoints with.

After initializing the sweep range as such, sweeping through the data, and running multiple FlexPDE scripts with the aid of the threading module simultaneously, the data is then collated by analyzing each output file and extracting the displacement value. The displacement is then saved in a list along with the polymer thickness and orientation that produced it. Each plot was specific to a material and the raw data from FlexPDE was plotted without any curve fitting or gradient ascent as detailed further below.

The program also utilizes the threading module, running 50 FlexPDE scripts simultaneously in a different thread for each iteration. This greatly improves performance through parallelization of the task in addition to the 2D simulation already being much faster than a 3D counterpart would be and as such, make running the code in a somewhat reasonable time possible. A brief summary of each code file is included in the flowcharts below to prevent breaking the flow of this report for the reader. However, for the full code, please refer to the appendices.

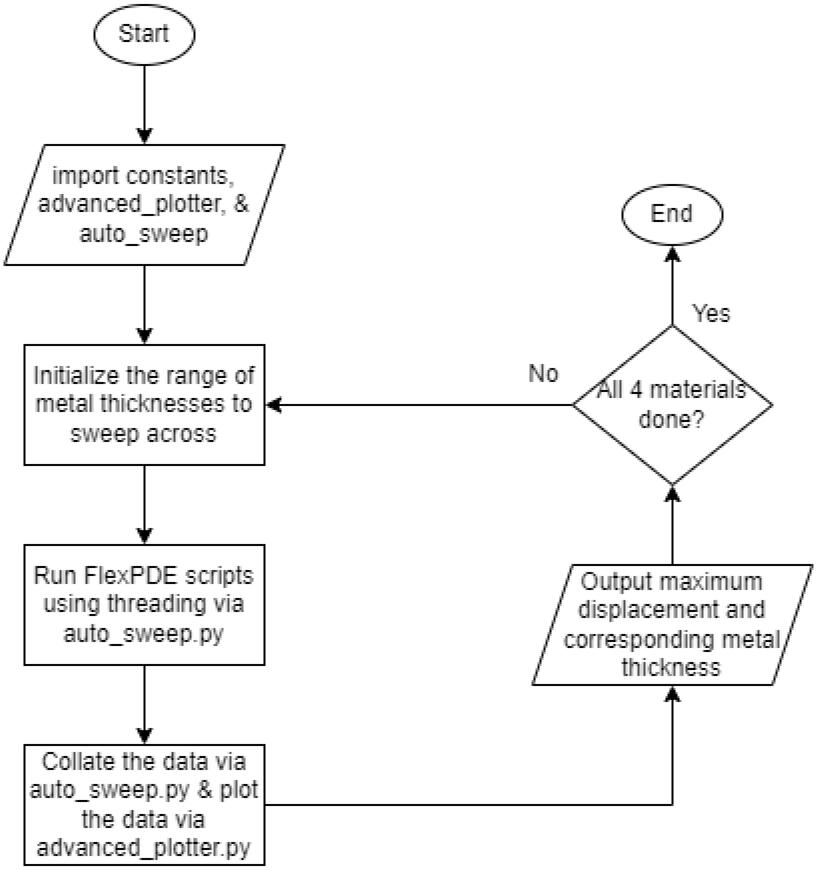

Start
import constants, advanced_plotter, & auto_sweep
Initialize the range of metal thicknesses to sweep across
Run FlexPDE scripts using threading via auto_sweep.py
Collate the data via auto_sweep.py & plot the data via advanced_plotter.py
Output maximum displacement and corresponding metal thickness
All 4 materials done?
No
Yes
End

Figure 5: Solver.py

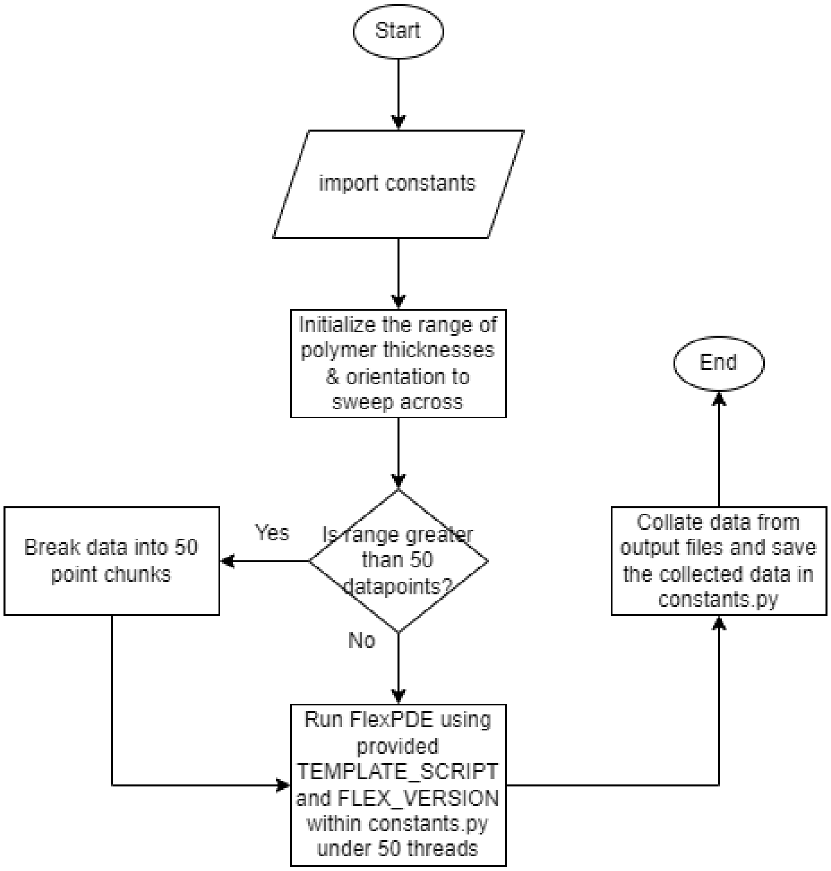

Start
import constants
Initialize the range of polymer thicknesses & orientation to sweep across
Is range greater than 50 datapoints?
Yes
Break data into 50 point chunks
No
Run FlexPDE using provided TEMPLATE_SCRIPT and FLEX_VERSION within constants.py under 50 threads
Collate data from output files and save the collected data in constants.py
End

Figure 6: auto_sweep.py

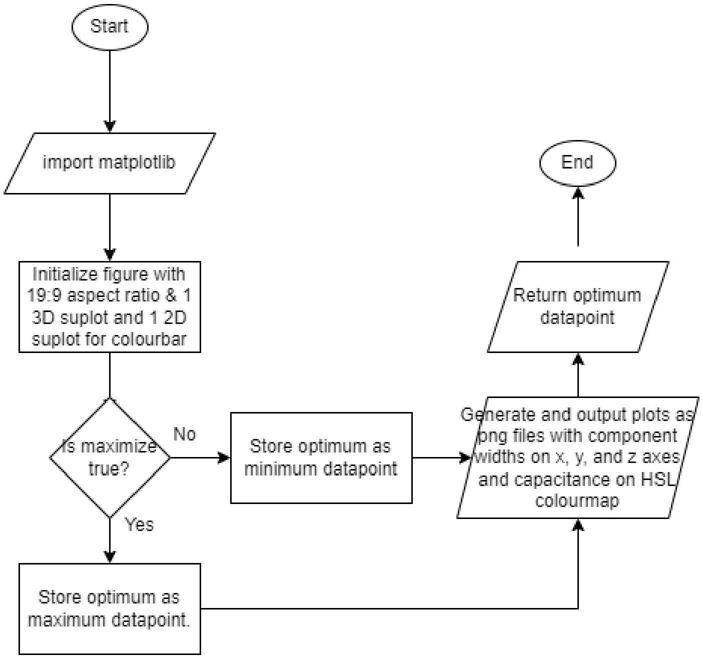

Start
import matplotlib
Initialize figure with 19:9 aspect ratio & 1 3D suplot and 1 2D suplot for colourbar
Is maximize true?
No
Store optimum as minimum datapoint
Yes
Store optimum as maximum datapoint.
Generate and output plots as png files with component widths on x, y, and z axes and capacitance on HSL colourmap
Return optimum datapoint
End

Figure 7: advanced_plotter.py

Optimization

As stated earlier, the time for this simulation to run took far longer than previous assignments and in particular was an issue with materials with higher Poisson's ratios being harder for FlexPDE to compute. This slowed down the compute times for both gold and vibranium significantly. As such, an optimization routine in the conventional sense was not employed to keep things somewhat simple and work within the limitations of the time. A grid plot was made and the maximum was chosen from the points. From here, the modal analysis script was run using the maximum to produce the resonant frequency for that particular configuration. This raw datapoint was output without further zooming in or any other technique such as gradient ascent or curve fitting. This is shown in the plots below for each material. Due to the nature of the raw datapoints, a maximum could be within a range of values surrounding what was found. However, the density of the mesh used allowed for it to be pinpointed to about 50 microns of precision in each dimension.

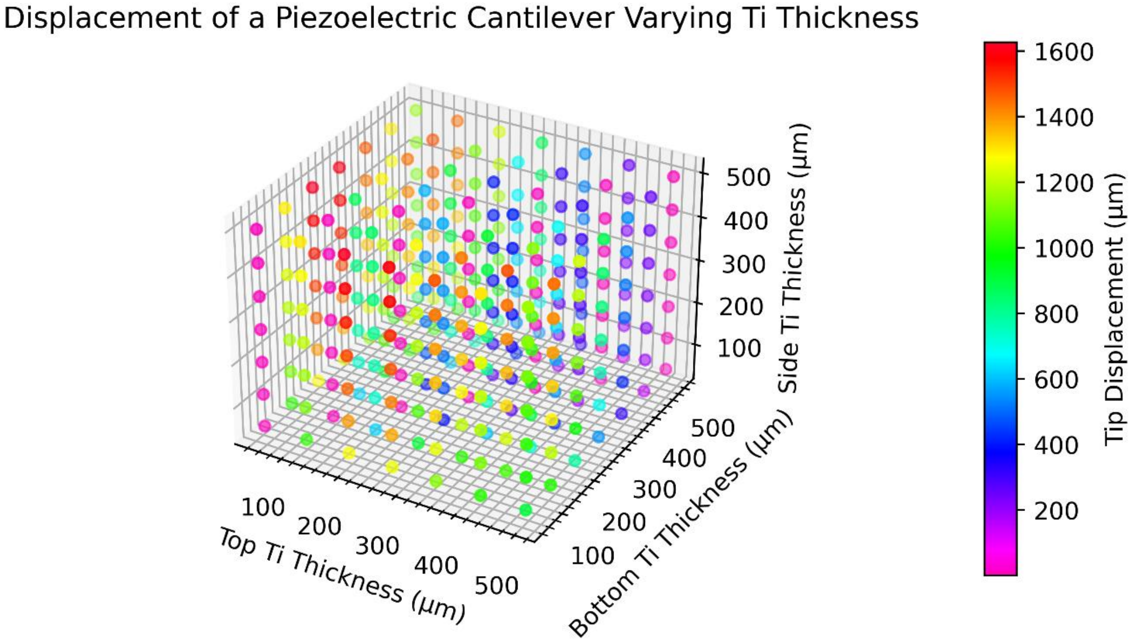


Figure 8: Max Displacement for Ti Thickness

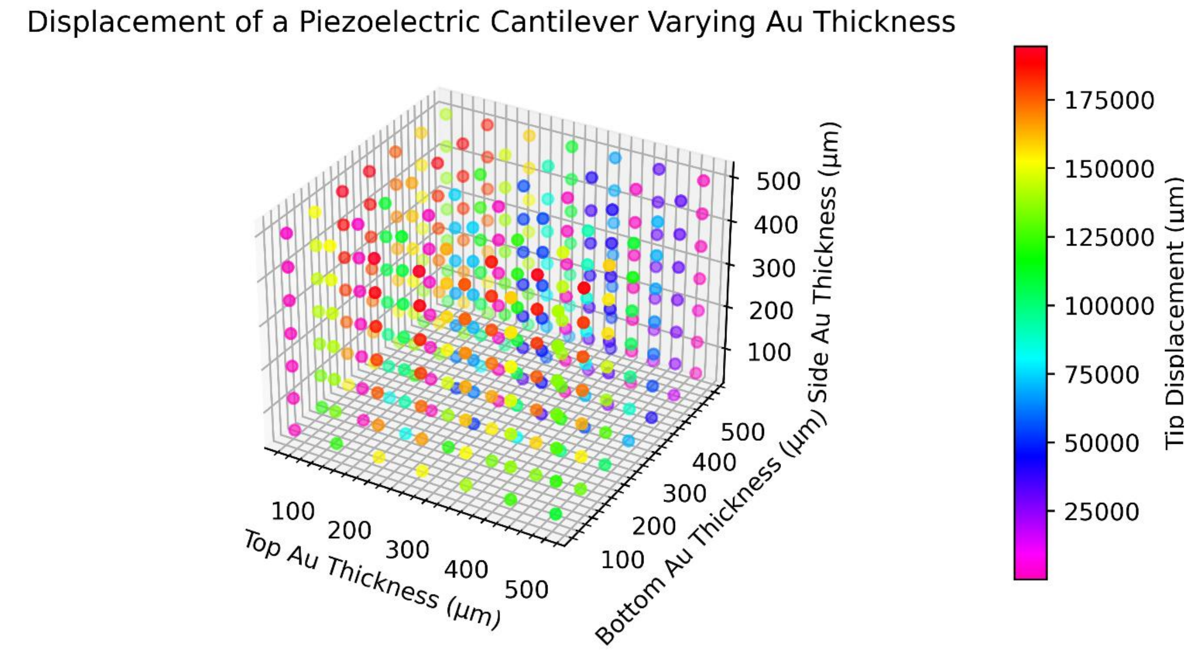


Figure 9: Max Displacement for Au Thickness

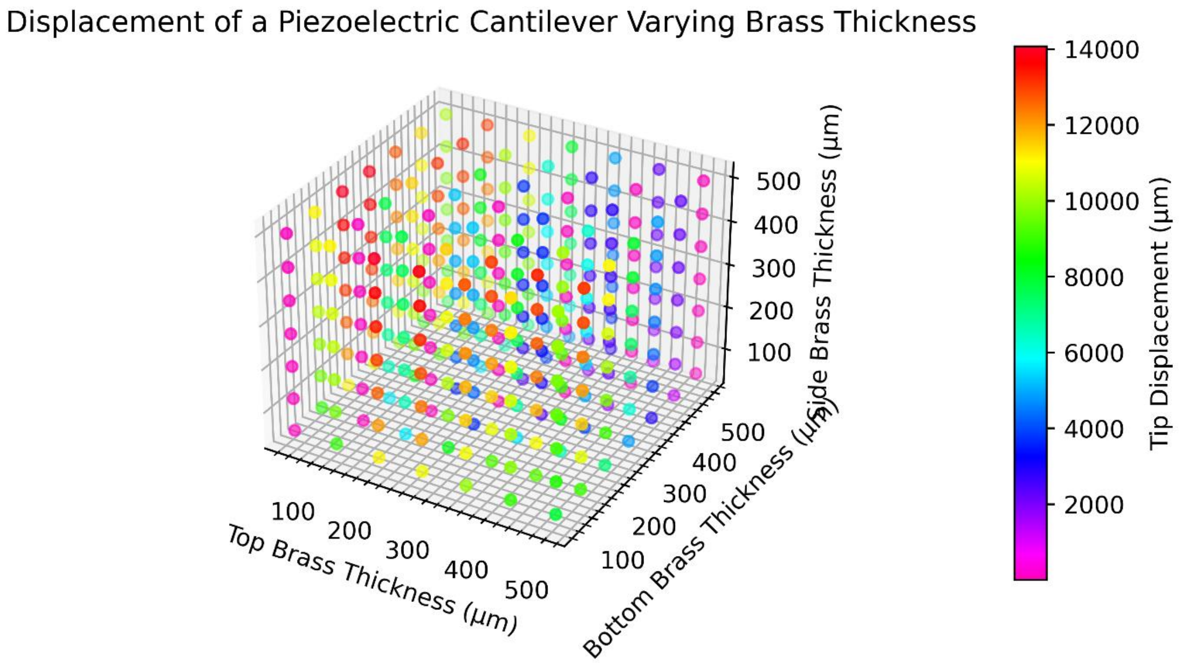


Figure 10: Max Displacement for Brass Thickness

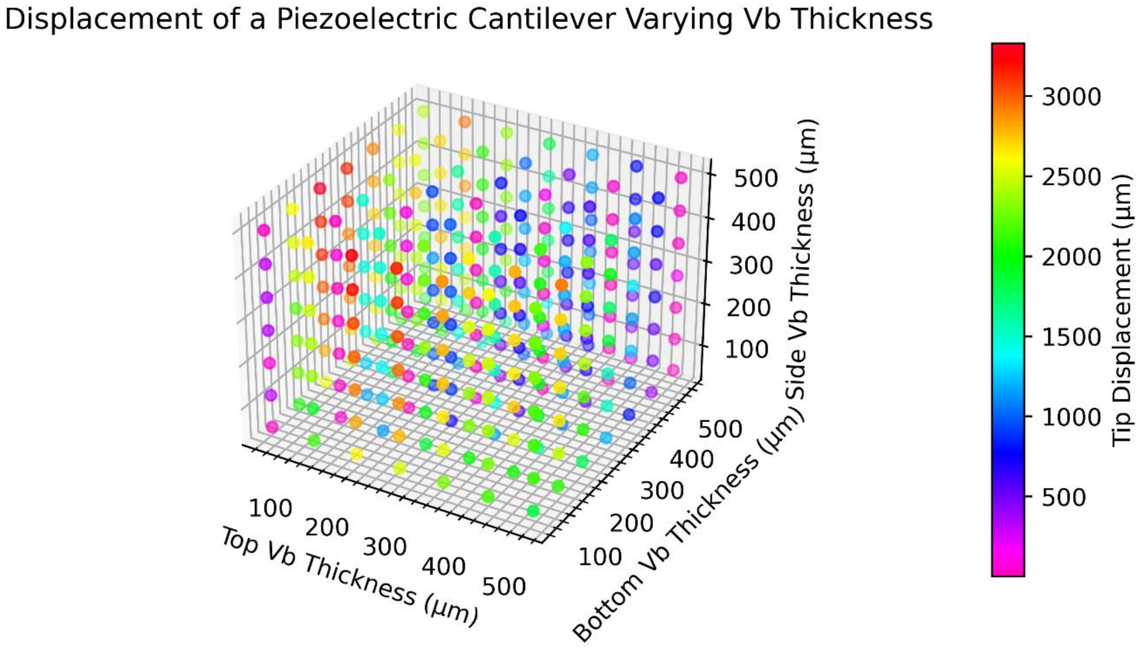


Figure 11: Max Displacement for Vibranium Thickness

Conclusion

To achieve the maximum displacement of such a device under the provided constraints, the optimal material properties were found to be using gold with a thickness of 275 microns on top, 50 microns on the bottom, and 500 microns on the side. The other results are also presented below, however, it should be noted that all designs would require cooling due to the high temperatures they reach. Moreover, some of the results should be taken with a grain of salt as the accuracy could be significantly improved with a higher ngrid but was not entirely implemented across the board due to time constraints.

```
--Ti Maximum--

t_metal_top: 200.0 (µm), t_metal_bottom: 50.0 (µm), t_metal_side: 500.0 (µm)

displacement: 1627.163 (µm), temp_junc: 41143.23 (°C), res_freq: 62025.72 (Hz)


--Au Maximum--

t_metal_top: 275.0 (µm), t_metal_bottom: 50.0 (µm), t_metal_side: 500.0 (µm)

displacement: 194464.0 (µm), temp_junc: 830749.4 (°C), res_freq: 31318.9 (Hz)


--Brass Maximum--

t_metal_top: 200.0 (µm), t_metal_bottom: 50.0 (µm), t_metal_side: 500.0 (µm)

displacement: 14072.0 (µm), temp_junc: 159273.5 (°C), res_freq: 43362.95 (Hz)


--Vb Maximum--

t_metal_top: 50.0 (µm), t_metal_bottom: 200.0 (µm), t_metal_side: 500.0 (µm)

displacement: 3329.5389999999998 (µm), temp_junc: 134325.9 (°C), res_freq: 211359.8 (Hz)
```

Thus this simulation is intended as a rough guideline to provoke further analysis. The model can also be further validated from these results as the optimum appears to be at some value with difference between the top and bottom thicknesses which is necessary to actuate larger amounts due to thermal strain and at the maximum possible thickness for the side. This elongates the beam and allows it to displace more. Thus, the final values can be assumed to be accurate to a degree. It should also be noted that while vibranium does not stand out too much amongst the other materials, it does have the highest resonant frequency which is expected as it is made to absorb kinetic energy and as such must be vibrating rapidly in order to dissipate them. While titanium did have a smaller displacement than vibranium, vibranium had the second smallest displacement due to its stiffness which is required to take and redistribute an impact. Thus, the tantalum alloy proposed to be a vibranium substitute could indeed be viable albeit not to the extent seen in movies.

It should be noted that certain compromises had to be made within this project. Most notable, is the rounding to 1 μm required due to the limitations imposed by FlexPDE. The maximum value was also identified through only using raw data without any further refinement. Moreover, this was also done in a 2D simulation and could have some differences to a 3D simulation due to warping of the shape, although for the most part negligible. Hence, while the simulation is not perfect, it can still be accepted to be usable within a reasonable amount of precision and can be used as a starting point for future reference and as an initial analysis with further analysis to be done down the line.

References

[1] M. Minnick, “ENGPHYS 2CM4 Design Project.” Apr. 03, 2023

[2] M. Minnick, "EP2CM4 TX Problem Set." Dec. 24, 2020

[3] “Properties: Copper Alloys - Brasses.” https://www.azom.com/properties.aspx?ArticleID=63 (accessed Apr. 17, 2023).

[4] “Gold - Physical, Mechanical, Thermal, and Electrical Properties.” https://www.azom.com/article.aspx?ArticleID=5147 (accessed Apr. 17, 2023).

[5] M. Mendez-Rosales, “ENGPHYS 2CM4 ● Design Assignment 4.” Feb. 17, 2023

[6] “Vibranium - Material Properties - Matmatch.” https://go.matmatch.com/vibranium (accessed Apr. 17, 2023).

[7] M. Minnick, "EP2CM4 T2 Notes - Heat Flow." Jan. 30, 2023

[8] M. Minnick, "EP2CM4 T3 Notes – E&M." Feb. 09, 2023

[9] M. Minnick, "EP2CM4 T4 Notes – Thermal Expansion & Piezoelectrics." Feb. 24, 2022

[10] M. Minnick, "EP2CM4 T5 Notes – Modal Analysis & Beam Resonance." March. 19, 2021

Appendices

Final Version of Full Code

FlexPDE Code

Steady State

```
TITLE 'Design Project - Steady State'     ! The Problem Identification

COORDINATES cartesian2 ("x", "z")         ! Coordinate System, 1D,2D,3D, etc

VARIABLES
V(threshold=1e-6)                         ! Voltage (V)
rho_free(threshold=1e-6)                  ! Free Charge Density (C/m^3)
Temp(threshold=1e-6)                      ! Temperature (C)
u(threshold=1e-6)                         ! Displacement in x (m)
w(threshold=1e-6)                         ! Displacement in z (m)

SELECT
ngrid = 20                                ! Method Controls

DEFINITIONS
! Dynamic
alpha_T                             ! Coefficient of Thermal Expansion (K^-1)
k                                   ! Thermal Conductivity (W/m K^-1)

epsilon_r                           ! Relative Permittivity (unitless ratio)
sigma_e                             ! Electrical Conductivity (S/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 d_13 d_15
d_31 d_33 d_35

! Stiffness Matrix (Pa^-1)
C_11 C_13 C_15
C_31 C_33 C_35
C_51 C_53 C_55

! Static
mag = 0.1*globalmax(magnitude(x, z))/globalmax(magnitude(u, w))
                        ! Magnification factor for plot (unitless ratio)

epsilon_0 = 8.85e-12          ! Permittivity of Free Space (F/m)

h_air = 200                   ! Convection Coefficient for Air (W/m^2 K^-1)

E_metal = 120e9               ! Young's Modulus for Metal (Pa)
nu_metal = 0.34               ! Poisson's Ratio for Metal (unitless ratio)

V_max = 1                     ! Electric Potential over Piezoelectric (V)
```

```
l_x = 2e-3                    ! Length of Piezoelectric (m)
t_piezo = 2e-4                ! Thickness of Piezoelectric (m)

Delta_T = Temp - 25           ! Temperature Difference (K)

T_inf = 25                    ! Temperature of Surroundings (C)

! Sweep
t_metal_top = 1e-4                  ! Thickness of Metal on Top (m)
t_metal_bottom = 2e-4               ! Thickness of Metal on Bottom (m)
t_metal_side = 1e-4                 ! Thickness of Metal on Free Side (m)

! Equations
E_field = -grad(V)                  ! Electric Field (V/m)
J = sigma_e*E_field                 ! Current Density (A/m^2)
q_dot = -k*grad(Temp)               ! Heat Flux for Conduction (W/m^2)
q_dotc = h_air*(T_inf - Temp)       ! Heat Flux for Convection (W/m^2)
q_dotvol = dot(J, E_field)          ! Volumetric Heat Generation (W/m^3)

E_x = xcomp(E_field)                ! Electric Field in x (V/m)
E_z = zcomp(E_field)                ! Electric Field in z (V/m)

! Strain Definitions from Displacements (unitless ratio)
epsilon_x = dx(u)
epsilon_z = dz(w)
gamma_xz = dx(w) + dz(u)

! Mechanical Strain (unitless ratio)
epsilon_xm = epsilon_x - alpha_T*Delta_T - (d_11*E_x + d_31*E_z)
epsilon_zm = epsilon_z - alpha_T*Delta_T - (d_13*E_x + d_33*E_z)
gamma_xzm = gamma_xz - (d_15*E_x + d_35*E_z)

! Hookes Law for Stresses (Pa)
s_x = C_11*epsilon_xm + C_13*epsilon_zm + C_15*gamma_xzm
s_z = C_31*epsilon_xm + C_33*epsilon_zm + C_35*gamma_xzm
s_xz = C_51*epsilon_xm + C_53*epsilon_zm + C_55*gamma_xzm

! Electric Displacement Field due to Piezoelectric Effect (C/m^2)
D_piezox = d_11*s_x + d_13*s_z + d_15*s_xz
D_piezoz = d_31*s_x + d_33*s_z + d_35*s_xz

D_piezo = vector(D_piezox, 0, D_piezoz)

! Total Electric Displacement Field (C/m^2)
D_field = epsilon_0*epsilon_r*E_field + D_piezo

INITIAL VALUES
V = V_max/2                         ! Voltage begins at half V_max
Temp = T_inf      ! Cantilever begins at same temperature as surroundings
rho_free = 0                        ! No free charge on cantilever at start
```

```
EQUATIONS                           ! PDEs; one for each variable
V: div(J) = 0
rho_free: div(D_field) = rho_free
u: dx(s_x) + dz(s_xz) = 0
w: dx(s_xz) + dz(s_z) = 0
Temp: q_dotvol = div(q_dot)

BOUNDARIES
      REGION "Metal"
                  alpha_T = 8.6e-6  ! Coefficient of Thermal Expansion (K^-1)
                  k = 17            ! Thermal Conductivity (W/m K^-1)

                  epsilon_r = 1e6   ! Relative Permittivity (unitless ratio)
                  sigma_e = 1/4.2e-7      ! Electrical Conductivity (S/m)

                  ! Piezoelectric Coupling Coefficients (m/V)
                  d_11 = 0 d_13 = 0 d_15 = 0
                  d_31 = 0 d_33 = 0 d_35 = 0

                  ! Stiffness Matrix (Pa^-1)
                  C_11 = E_metal*(1 - nu_metal)/((1 + nu_metal)*(1 -
2*nu_metal)) C_13 =  E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal)) C_15
= 0
                  C_31 = E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal))
                              C_33 = C_11       C_35 = 0
                  C_51 = 0    C_53 = 0    C_55 = E_metal/(2*(1 + nu_metal))

                  START (0, 0)
                        load(Temp) = q_dotc     ! Set up convection on all
                                                free sides
                  LINE TO (l_x + t_metal_side, 0)
                  LINE TO (l_x + t_metal_side, t_piezo + t_metal_top +
t_metal_bottom)
                  LINE TO (0, t_piezo + t_metal_top + t_metal_bottom)
                                    ! Hold one side fixed for cantilever
                        value(V) = V_max  ! Set V to V_max on top contact
                        value(rho_free) = 0
                                  ! Set free charge density to zero on contact
                        load(Temp) = 0    ! Thermally insulate fixed side
                        value(u) = 0
                        value(w) = 0
                  LINE TO (0, t_piezo/2 + t_metal_bottom)
                              ! Hold one side fixed for cantilever, insulate
fixed side, and set free charge density to zero
                        value(V) = 0      ! Set V to 0 on bottom contact
                        value(rho_free) = 0
                              ! Set free charge density to zero on contact
                        load(Temp) = 0    ! Thermally insulate fixed side
                        value(u) = 0
                        value(w) = 0
                  LINE TO CLOSE

      REGION "Piezoelectric"
                  alpha_T = 1.15e-6 ! Coefficient of Thermal Expansion (K^-1)
                  k = 7.7           ! Thermal Conductivity (W/m K^-1)

                  epsilon_r = 4.68  ! Relative Permittivity (unitless ratio)
                  sigma_e = 1/10e5  ! Electrical Conductivity (S/m)

                  ! Piezoelectric Coupling Coefficients (m/V)
                  d_11 = 2e-11      d_13 = 0           d_15 = 0
                  d_31 = 4.5e-11 d_33 = 7e-11 d_35 = 0

                  ! Stiffness Matrix (Pa^-1)
                  C_11 = 8.66766243465273e10 C_13 = 1.12023898431665e10 C_15
= 0
                  C_31 = C_13  C_33 = 1.02688573562360e11 C_35 = 0
                  C_51 = 0     C_53 = 0         C_55 = 5.78034682080925e10

                  START (0, t_metal_bottom)
                  LINE TO (l_x, t_metal_bottom)
                  LINE TO (l_x, t_metal_bottom + t_piezo)
                  LINE TO (0, t_metal_bottom + t_piezo)
                              ! Hold one side fixed for cantilever
                        load(Temp) = 0    ! Thermally insulate fixed side
                        value(u) = 0
                        value(w) = 0
                  LINE TO CLOSE

PLOTS                               ! Save result displays
      grid(x + mag*u, z + mag*w)
      CONTOUR(V) painted
      contour(rho_free) painted
      vector(J) norm
      vector(E_field) norm
      vector(D_field) norm
      contour(Temp) painted
      vector(q_dot) norm
SUMMARY export file="stdout.txt"
      report val(w, l_x + t_metal_side, t_metal_bottom + t_piezo/2) as
"Delta_tip "
                              ! Export Transverse Tip Displacement (m)
      report val(Temp, l_x, t_metal_bottom + t_piezo/2) as "T_junc "
                              ! Export Tip Junction Temperature (m)
END
```

Resonance

```
TITLE 'Design Project - Resonance'        ! The Problem Identification

COORDINATES cartesian2 ("x", "z")         ! Coordinate System, 1D,2D,3D, etc

VARIABLES
u(threshold=1e-6)                         ! Displacement in x (m)
w(threshold=1e-6)                         ! Displacement in z (m)

SELECT                                    ! Method Controls
ngrid = 20
modes = 1

DEFINITIONS
! Dynamic
alpha_T                             ! Coefficient of Thermal Expansion (K^-1)

rho                                 ! Density (kg/m^3)

E_x                                 ! Electric Field in x (V/m)
E_z                                 ! Electric Field in z (V/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 d_13 d_15
d_31 d_33 d_35

! Stiffness Matrix (Pa^-1)
C_11 C_13 C_15
C_31 C_33 C_35
C_51 C_53 C_55

! Static
mag = 0.1*globalmax(magnitude(x, z))/globalmax(magnitude(u, w))         !
Magnification factor for plot (unitless ratio)

epsilon_0 = 8.85e-12          ! Permittivity of Free Space (F/m)

E_metal = 120e9               ! Young's Modulus for Metal (Pa)
nu_metal = 0.34               ! Poisson's Ratio for Metal (unitless ratio)

V_max = 1                     ! Electric Potential over Piezoelectric (V)

l_x = 2e-3                    ! Length of Piezoelectric (m)
t_piezo = 2e-4                ! Thickness of Piezoelectric (m)

Delta_T = 25                  ! Temperature Difference (K)

! Sweep
t_metal_top = 1e-4            ! Thickness of Metal on Top (m)
t_metal_bottom = 2e-4         ! Thickness of Metal on Bottom (m)
```

```
t_metal_side = 1e-4           ! Thickness of Metal on Free Side (m)

! Equations
! Strain Definitions from Displacements (unitless ratio)
epsilon_x = dx(u)
epsilon_z = dz(w)
gamma_xz = dx(w) + dz(u)

! Mechanical Strain (unitless ratio)
epsilon_xm = epsilon_x - alpha_T*Delta_T - (d_11*E_x + d_31*E_z)
epsilon_zm = epsilon_z - alpha_T*Delta_T - (d_13*E_x + d_33*E_z)
gamma_xzm = gamma_xz - (d_15*E_x + d_35*E_z)

! Hookes Law for Stresses (Pa)
s_x = C_11*epsilon_xm + C_13*epsilon_zm + C_15*gamma_xzm
s_z = C_31*epsilon_xm + C_33*epsilon_zm + C_35*gamma_xzm
s_xz = C_51*epsilon_xm + C_53*epsilon_zm + C_55*gamma_xzm

EQUATIONS                     ! PDEs; one for each variable
u: dx(s_x) + dz(s_xz) = -rho*lambda*u
w: dx(s_xz) + dz(s_z) = -rho*lambda*w

BOUNDARIES
      REGION "Metal"
                  alpha_T = 8.6e-6  ! Coefficient of Thermal Expansion (K^-1)

                  rho = 4500        ! Density (kg/m^3)

                  E_x = 0           ! Electric Field in x (V/m)
                  E_z = 0           ! Electric Field in z (V/m)

                  ! Piezoelectric Coupling Coefficients (m/V)
                  d_11 = 0 d_13 = 0 d_15 = 0
                  d_31 = 0 d_33 = 0 d_35 = 0

                  ! Stiffness Matrix (Pa^-1)
                  C_11 = E_metal*(1 - nu_metal)/((1 + nu_metal)*(1 -
2*nu_metal)) C_13 =  E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal)) C_15
= 0
                  C_31 = E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal))
                              C_33 = C_11       C_35 = 0
                  C_51 = 0    C_53 = 0     C_55 = E_metal/(2*(1 + nu_metal))

                  START (0, 0)
                  LINE TO (l_x + t_metal_side, 0)
                  LINE TO (l_x + t_metal_side, t_piezo + t_metal_top +
t_metal_bottom)
                  LINE TO (0, t_piezo + t_metal_top + t_metal_bottom)
                        ! Hold one side fixed for cantilever
                        value(u) = 0
                        value(w) = 0
                  LINE TO CLOSE

      REGION "Piezoelectric"
                  alpha_T = 1.15e-6 ! Coefficient of Thermal Expansion (K^-1)

                  rho = 2650        ! Density (kg/m^3)

                  E_x = 0           ! Electric Field in x (V/m)
                  E_z = V_max/t_piezo     ! Electric Field in z (V/m)

                  ! Piezoelectric Coupling Coefficients (m/V)
                  d_11 = 2e-11      d_13 = 0           d_15 = 0
                  d_31 = 4.5e-11 d_33 = 7e-11 d_35 = 0

                  ! Stiffness Matrix (Pa^-1)
                  C_11 = 8.66766243465273e10 C_13 = 1.12023898431665e10 C_15
= 0
                  C_31 = C_13       C_33 = 1.02688573562360e11 C_35 = 0
                  C_51 = 0          C_53 = 0    C_55 = 5.78034682080925e10

                  START (0, t_metal_bottom)
                  LINE TO (l_x, t_metal_bottom)
                  LINE TO (l_x, t_metal_bottom + t_piezo)
                  LINE TO (0, t_metal_bottom + t_piezo)
                              ! Hold one side fixed for cantilever
                        value(u) = 0
                        value(w) = 0
                  LINE TO CLOSE

PLOTS                               ! Save result displays
      grid(x + mag*u, z + mag*w)
SUMMARY export file="stdout.txt"
      report sqrt(lambda)/(2*pi) as "f "  ! Export Resonant Frequency (Hz)
END
```

Python code

Constants.py

```
"""Global constants and variables to be held for other programs to access.

   Inputs:
            auto_sweep.py: Stores output of data in global variables

   Outputs:
            auto_sweep.py: Provides TEMPLATE_SCRIPT and FLEX_VERSION
for use in running FlexPDE
 solver.py: Provides data output from global variables and inital sweep
parameters
"""

# Template FlexPDE script to be modified for sweep
TEMPLATE_SCRIPT = """
TITLE 'Design Project - Steady State'
                                    ! The Problem Identification

COORDINATES cartesian2 ("x", "z")
                                    ! Coordinate System, 1D,2D,3D, etc

VARIABLES
V(threshold=1e-6)
                                          ! Voltage (V)

rho_free(threshold=1e-6)
                                          ! Free Charge Density (C/m^3)

Temp(threshold=1e-6)
                                          ! Temperature (C)

u(threshold=1e-6)
                                          ! Displacement in x (m)

w(threshold=1e-6)
                                          ! Displacement in z (m)

SELECT
ngrid = 20
                                                ! Method Controls

DEFINITIONS
! Dynamic
alpha_T
                                                      ! Coefficient of

Thermal Expansion (K^-1)
k
                                                      ! Thermal

Conductivity (W/m K^-1)
```

```
epsilon_r
                                                ! Relative Permittivity
(unitless ratio)
sigma_e
                                                      ! Electrical
Conductivity (S/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 d_13 d_15
d_31 d_33 d_35

! Stiffness Matrix (Pa^-1)
C_11 C_13 C_15
C_31 C_33 C_35
C_51 C_53 C_55

! Static
mag = 0.1*globalmax(magnitude(x, z))/globalmax(magnitude(u, w))
                              ! Magnification factor for plot (unitless
ratio)

epsilon_0 = 8.85e-12
                                          ! Permittivity of Free Space
(F/m)

h_air = 200
                                                ! Convection Coefficient
for Air (W/m^2 K^-1)

E_metal = {0}
                                                ! Young's Modulus for Metal
(Pa)
nu_metal = {1}
                                                ! Poisson's Ratio for Metal
(unitless ratio)

V_max = 1
                                                ! Electric Potential over
Piezoelectric (V)

l_x = 2e-3
                                                ! Length of Piezoelectric
(m)
t_piezo = 2e-4
                                                ! Thickness of
Piezoelectric (m)
```

```
Delta_T = Temp - 25
                                                ! Temperature Difference
(K)

T_inf = 25
                                                ! Temperature of
Surroundings (C)

! Sweep
t_metal_top = {2}
                                          ! Thickness of Metal on Top (m)
t_metal_bottom = {3}
                                          ! Thickness of Metal on Bottom
(m)
t_metal_side = {4}
                                                ! Thickness of Metal on
Free Side (m)

! Equations
E_field = -grad(V)
                                                ! Electric Field (V/m)
J = sigma_e*E_field
                                                ! Current Density (A/m^2)
q_dot = -k*grad(Temp)
                                          ! Heat Flux for Conduction
(W/m^2)
q_dotc = h_air*(T_inf - Temp)
                                    ! Heat Flux for Convection (W/m^2)
q_dotvol = dot(J, E_field)
                                          ! Volumetric Heat Generation
(W/m^3)

E_x = xcomp(E_field)
                                          ! Electric Field in x (V/m)
E_z = zcomp(E_field)
                                          ! Electric Field in z (V/m)

! Strain Definitions from Displacements (unitless ratio)
epsilon_x = dx(u)
epsilon_z = dz(w)
gamma_xz = dx(w) + dz(u)

! Mechanical Strain (unitless ratio)
epsilon_xm = epsilon_x - alpha_T*Delta_T - (d_11*E_x + d_31*E_z)
epsilon_zm = epsilon_z - alpha_T*Delta_T - (d_13*E_x + d_33*E_z)
gamma_xzm = gamma_xz - (d_15*E_x + d_35*E_z)

! Hookes Law for Stresses (Pa)
```

```
s_x = C_11*epsilon_xm + C_13*epsilon_zm + C_15*gamma_xzm
s_z = C_31*epsilon_xm + C_33*epsilon_zm + C_35*gamma_xzm
s_xz = C_51*epsilon_xm + C_53*epsilon_zm + C_55*gamma_xzm

! Electric Displacement Field due to Piezoelectric Effect (C/m^2)
D_piezox = d_11*s_x + d_13*s_z + d_15*s_xz
D_piezoz = d_31*s_x + d_33*s_z + d_35*s_xz

D_piezo = vector(D_piezox, 0, D_piezoz)

! Total Electric Displacement Field (C/m^2)
D_field = epsilon_0*epsilon_r*E_field + D_piezo

INITIAL VALUES
V = V_max/2
                                                ! Voltage begins at half
V_max
Temp = T_inf
                                                ! Cantilever begins at same
temperature as surroundings
rho_free = 0
                                                ! No free charge on
cantilever at start

EQUATIONS
                                          ! PDEs; one for each variable
V: div(J) = 0
rho_free: div(D_field) = rho_free
u: dx(s_x) + dz(s_xz) = 0
w: dx(s_xz) + dz(s_z) = 0
Temp: q_dotvol = div(q_dot)

BOUNDARIES
REGION "Metal"
alpha_T = {5}
                              ! Coefficient of Thermal Expansion (K^-1)
k = {6}
                                    ! Thermal Conductivity (W/m K^-1)

epsilon_r = 1e6
                              ! Relative Permittivity (unitless ratio)
sigma_e = 1/{7}
                              ! Electrical Conductivity (S/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 = 0 d_13 = 0 d_15 = 0
d_31 = 0 d_33 = 0 d_35 = 0
```

```
! Stiffness Matrix (Pa^-1)
C_11 = E_metal*(1 - nu_metal)/((1 + nu_metal)*(1 - 2*nu_metal)) C_13 =
E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal)) C_15 = 0
C_31 = E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal))       C_33 = C_11
                                                                   C_35 = 0
C_51 = 0
            C_53 = 0
               C_55 = E_metal/(2*(1 + nu_metal))

START (0, 0)
load(Temp) = q_dotc
                        ! Set up convection on all free sides
LINE TO (l_x + t_metal_side, 0)
LINE TO (l_x + t_metal_side, t_piezo + t_metal_top + t_metal_bottom)
LINE TO (0, t_piezo + t_metal_top + t_metal_bottom)
            ! Hold one side fixed for cantilever
value(V) = V_max
                  ! Set V to V_max on top contact
value(rho_free) = 0
                        ! Set free charge density to zero on contact
load(Temp) = 0
                        ! Thermally insulate fixed side
value(u) = 0
value(w) = 0
LINE TO (0, t_piezo/2 + t_metal_bottom)
                  ! Hold one side fixed for cantilever
value(V) = 0
                        ! Set V to 0 on bottom contact
value(rho_free) = 0
                        ! Set free charge density to zero on contact
load(Temp) = 0
                        ! Thermally insulate fixed side
value(u) = 0
value(w) = 0
LINE TO CLOSE

REGION "Piezoelectric"
alpha_T = 1.15e-6
                        ! Coefficient of Thermal Expansion (K^-1)
k = 7.7
                                    ! Thermal Conductivity (W/m K^-1)

epsilon_r = 4.68
                        ! Relative Permittivity (unitless ratio)
sigma_e = 1/10e5
                        ! Electrical Conductivity (S/m)

! Piezoelectric Coupling Coefficients (m/V)
```

```
d_11 = 2e-11   d_13 = 0     d_15 = 0
d_31 = 4.5e-11 d_33 = 7e-11 d_35 = 0

! Stiffness Matrix (Pa^-1)
C_11 = 8.66766243465273e10 C_13 = 1.12023898431665e10 C_15 = 0
C_31 = C_13                      C_33 = 1.02688573562360e11 C_35 = 0
C_51 = 0                         C_53 = 0                           C_55 =
5.78034682080925e10

START (0, t_metal_bottom)
LINE TO (l_x, t_metal_bottom)
LINE TO (l_x, t_metal_bottom + t_piezo)
LINE TO (0, t_metal_bottom + t_piezo)
                  ! Hold one side fixed for cantilever
load(Temp) = 0
                        ! Thermally insulate fixed side
value(u) = 0
value(w) = 0
LINE TO CLOSE

PLOTS
                                                ! Save result displays
grid(x + mag*u, z + mag*w)
SUMMARY export file="output_{8}.txt"
report val(w, l_x + t_metal_side, t_metal_bottom + t_piezo/2) as "Delta_tip "
      ! Export Transverse Tip Displacement (m)
report val(Temp, l_x, t_metal_bottom + t_piezo/2) as "T_junc "
                        ! Export Tip Junction Temperature (m)
END
"""

# Second Template FlexPDE script to be modified for sweep
SECONDARY_TEMPLATE_SCRIPT = """
TITLE 'Design Project - Resonance'
      ! The Problem Identification

COORDINATES cartesian2 ("x", "z")
      ! Coordinate System, 1D,2D,3D, etc

VARIABLES
u(threshold=1e-6)
            ! Displacement in x (m)
w(threshold=1e-6)
            ! Displacement in z (m)

SELECT
                  ! Method Controls
ngrid = 20
```

```
modes = 1

DEFINITIONS
! Dynamic
alpha_T
                        ! Coefficient of Thermal Expansion (K^-1)

rho
                        ! Density (kg/m^3)

E_x
                        ! Electric Field in x (V/m)
E_z
                        ! Electric Field in z (V/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 d_13 d_15
d_31 d_33 d_35

! Stiffness Matrix (Pa^-1)
C_11 C_13 C_15
C_31 C_33 C_35
C_51 C_53 C_55

! Static
mag = 0.1*globalmax(magnitude(x, z))/globalmax(magnitude(u, w))
      ! Magnification factor for plot (unitless ratio)

epsilon_0 = 8.85e-12
            ! Permittivity of Free Space (F/m)

E_metal = {0}
                  ! Young's Modulus for Metal (Pa)
nu_metal = {1}
                  ! Poisson's Ratio for Metal (unitless ratio)

V_max = 1
                  ! Electric Potential over Piezoelectric (V)

l_x = 2e-3
                  ! Length of Piezoelectric (m)
t_piezo = 2e-4
                  ! Thickness of Piezoelectric (m)

Delta_T = 25
                  ! Temperature Difference (K)

! Sweep
```

```
t_metal_top = {2}
            ! Thickness of Metal on Top (m)
t_metal_bottom = {3}
            ! Thickness of Metal on Bottom (m)
t_metal_side = {4}
                  ! Thickness of Metal on Free Side (m)

! Equations
! Strain Definitions from Displacements (unitless ratio)
epsilon_x = dx(u)
epsilon_z = dz(w)
gamma_xz = dx(w) + dz(u)

! Mechanical Strain (unitless ratio)
epsilon_xm = epsilon_x - alpha_T*Delta_T - (d_11*E_x + d_31*E_z)
epsilon_zm = epsilon_z - alpha_T*Delta_T - (d_13*E_x + d_33*E_z)
gamma_xzm = gamma_xz - (d_15*E_x + d_35*E_z)

! Hookes Law for Stresses (Pa)
s_x = C_11*epsilon_xm + C_13*epsilon_zm + C_15*gamma_xzm
s_z = C_31*epsilon_xm + C_33*epsilon_zm + C_35*gamma_xzm
s_xz = C_51*epsilon_xm + C_53*epsilon_zm + C_55*gamma_xzm

EQUATIONS
            ! PDEs; one for each variable
u: dx(s_x) + dz(s_xz) = -rho*lambda*u
w: dx(s_xz) + dz(s_z) = -rho*lambda*w

BOUNDARIES
REGION "Metal"
alpha_T = {5}
      ! Coefficient of Thermal Expansion (K^-1)

rho = {6}
      ! Density (kg/m^3)

E_x = 0
      ! Electric Field in x (V/m)
E_z = 0
      ! Electric Field in z (V/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 = 0 d_13 = 0 d_15 = 0
d_31 = 0 d_33 = 0 d_35 = 0

! Stiffness Matrix (Pa^-1)
C_11 = E_metal*(1 - nu_metal)/((1 + nu_metal)*(1 - 2*nu_metal)) C_13 =
E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal)) C_15 = 0
```

```
C_31 = E_metal*nu_metal/((1 + nu_metal)*(1 - 2*nu_metal))        C_33 = C_11
                                                                    C_35 = 0
C_51 = 0
            C_53 = 0
               C_55 = E_metal/(2*(1 + nu_metal))

START (0, 0)
LINE TO (l_x + t_metal_side, 0)
LINE TO (l_x + t_metal_side, t_piezo + t_metal_top + t_metal_bottom)
LINE TO (0, t_piezo + t_metal_top + t_metal_bottom)          ! Hold one
side fixed for cantilever
value(u) = 0
value(w) = 0
LINE TO CLOSE

REGION "Piezoelectric"
alpha_T = 1.15e-6                                                   !
Coefficient of Thermal Expansion (K^-1)

rho = 2650
      ! Density (kg/m^3)

E_x = 0
      ! Electric Field in x (V/m)
E_z = V_max/t_piezo
      ! Electric Field in z (V/m)

! Piezoelectric Coupling Coefficients (m/V)
d_11 = 2e-11   d_13 = 0      d_15 = 0
d_31 = 4.5e-11 d_33 = 7e-11 d_35 = 0

! Stiffness Matrix (Pa^-1)
C_11 = 8.66766243465273e10 C_13 = 1.12023898431665e10 C_15 = 0
C_31 = C_13                      C_33 = 1.02688573562360e11 C_35 = 0
C_51 = 0                         C_53 = 0                        C_55 =
5.78034682080925e10

START (0, t_metal_bottom)
LINE TO (l_x, t_metal_bottom)
LINE TO (l_x, t_metal_bottom + t_piezo)
LINE TO (0, t_metal_bottom + t_piezo)                        ! Hold one
side fixed for cantilever
value(u) = 0
value(w) = 0
LINE TO CLOSE

PLOTS
                  ! Save result displays
```

```
grid(x + mag*u, z + mag*w)
SUMMARY export file="frequency.txt"
report sqrt(lambda)/(2*pi) as "f "
      ! Export Resonant Frequency (Hz)
END
"""

# Minimum and maximum thickness to begin sweep range
MIN_THICK = 5e-5
MAX_THICK = 5e-4

# Initialize empty lists to store width range to sweep and results of
sweep
sweep_range = []
displacement, temp_junc, t_metal_top, t_metal_bottom, t_metal_side = [], [],
[], [], []

# Metal Layer Properties
E_LS = [120e9, 79e9, 100e9, 300e9]
NU_LS = [0.34, 0.42, 0.35, 0.49]
ALPHA_T_LS = [8.6e-6, 4.2e-5, 1.7e-5, 9.5e-6]
K_LS = [17, 310, 150, 54]
SIGMA_E_LS = [4.2e-7, 2.2e-8, 1.1e-7, 1.3e-7]
RHO_LS = [4500, 19300, 8500, 497]

# List of all metals
MATERIAL_LS = ["Ti", "Au", "Brass", "Vb"]

# Path where your FlexPDE executable is installed
# (or just name of executable if added to PATH correctly)
FLEX_VERSION = "FlexPDE6s"

# Ensuring the script is only imported into other programs as intended
if __name__ == "__main__":
    print("This script is only meant to be implemented as a module, not
run independently.")
```

auto_sweep.py

```
"""Program containing all necessary functions to run flex,
   sweep through data, and organize results. Held for other programs to
access.

   Inputs:
         constants.py: Loads TEMPLATE_SCRIPT and FLEX_VERSION for use in
running FlexPDE
         solver.py: Global functions are called from within solver.py with
any parameters necessary passed to them
```

```
         output*.txt: Takes output data from FlexPDE and collates it

   Outputs:
         constants.py: Saves output data in constants.py to facilitate
sharing across multiple scripts
         output*.txt: Directs FlexPDE to store output data in text files
"""

# Imports
import constants as ct
from threading import Thread
from itertools import groupby
from numpy import linspace
from subprocess import run
from typing import List

# Initialize empty list to keep track of how many processes are alive at once
threads = list()

# Function Definitions


def initialize_range(*args: List[int], points=7):
    """Initializes given a list of upper and lower bounds to sweep from for
       each value to be varied. Creates a specified number of
       sweep points (default is 7).

       Args:
           *args (List[List[int]]): A list of lists, each list within
           containing an upper and lower bound to be unpacked and swept
           through as a range for its respective variable
           points (int): Defualt value is 7; Number of points to make for
                         sweeping through range for each variable

       Returns:
           None
    """

    # Set up range for each variable provided with bounds
    var_ranges = [linspace(i[0], i[1], points) for i in args]

    # Place each valid value in range in order and grouped for running in a
FlexPDE script
    # Round all values to 6 decimal places and remove any duplicates after
rounding
    rounded_vars = [[round(i, 6), round(j, 6), round(
        k, 6)] for i in var_ranges[0] for j in var_ranges[1] for k in
var_ranges[2]]

    ct.sweep_range = [i for i, _ in groupby(rounded_vars)]


def run_flex(name: str, *args: List[str]):
    """Runs FlexPDE given a name for the script and any arguments to pass to
       the template script located in constants. Outputs the file name and
       returncode. Saves and runs the FlexPDE script in the working
directory.

       Args:
           name (str): A name for the FlexPDE script excluding the file
extension
           *args (List[str]): Parameters to be passed onto the script in
constants
                        through the format method and in the order they were
given

       Returns:
           None
    """

    # Open FlexPDE script and write formatted template
    with open(f"{name}.pde", "w") as stdin:
        print(ct.TEMPLATE_SCRIPT.format(*args), file=stdin)

    # Adjust timeout according to computing power
    # (if scripts return with timeout, increase timeout)
    completed = run([ct.FLEX_VERSION, "-S", f"{name}.pde"], timeout=1000)

    # Output runcode of file to confirm successful completion
    print(f"File {name} returned {completed.returncode}")


def sweep(material: str):
    """Sweeps through all values present in the pre-defined sweep range by
       instantiating a new thread for each datapoint and running the
       corresponding number of FlexPDE instances to produce the datapoints.
       Outputs when a new thread has started and when all threads have
completed.

       Args:
           None
       Returns:
           None
    """

    # Set maximum number of threads to run at once, change depending on CPU
core count
    chunk_size = 50

    # Get length of range to break into chunks
    sweep_length = len(ct.sweep_range)

    # Break range into manageable chunks
    sweep_blocks = [ct.sweep_range[i:i + chunk_size]
                    for i in range(0, sweep_length, chunk_size)]

    # Taking all items in the sweep range and indexes to be each
    # passed into a thread with the target set to run_flex()
    for (i, chunk) in enumerate(sweep_blocks):
        for (j, t_list) in enumerate(chunk):
            identifier = j + i*chunk_size

            if material == "Ti":
                t = Thread(target=run_flex, args=(f"script_{identifier}",
ct.E_LS[0], ct.NU_LS[0], t_list[
                           0], t_list[1], t_list[2],  ct.ALPHA_T_LS[0],
ct.K_LS[0], ct.SIGMA_E_LS[0], identifier,))
            elif material == "Au":
                t = Thread(target=run_flex, args=(f"script_{identifier}",
ct.E_LS[1], ct.NU_LS[1], t_list[
                           0], t_list[1], t_list[2],  ct.ALPHA_T_LS[1],
ct.K_LS[1], ct.SIGMA_E_LS[1], identifier,))
            elif material == "Brass":
                t = Thread(target=run_flex, args=(f"script_{identifier}",
ct.E_LS[2], ct.NU_LS[2], t_list[
                           0], t_list[1], t_list[2],  ct.ALPHA_T_LS[2],
ct.K_LS[2], ct.SIGMA_E_LS[2], identifier,))
            elif material == "Vb":
                t = Thread(target=run_flex, args=(f"script_{identifier}",
ct.E_LS[3], ct.NU_LS[3], t_list[
                           0], t_list[1], t_list[2],  ct.ALPHA_T_LS[3],
ct.K_LS[3], ct.SIGMA_E_LS[3], identifier,))

            threads.append(t)
            t.start()
            print(f"Thread script_{identifier} Started")
        # Wait for all threads to finish before moving on to next step
        # to prevent FileNotFound errors
        while True:
            alive = [thread.is_alive() for thread in threads]
            if not any(alive):
                print("All threads complete!")
                break


def collate_data():
    """Collates all data from the output files produced by FlexPDE
       and saves them in blank lists stored in constants.

       Args:
           None

       Returns:
           None
    """

    # Re-initalize output variables as empty lists to ensure
    # previous data from other iterations is erased
    ct.displacement, ct.temp_junc, ct.t_metal_top, ct.t_metal_bottom,
ct.t_metal_side = [], [], [], [], []

    # Taking all items in the sweep range and indexes to view output files
    # produced by FlexPDE
    for (i, t_list) in enumerate(ct.sweep_range):
        with open(f"output_{i}.txt", "r") as stdin:
            data = stdin.readlines()

            d = abs(float(data[6].split(" = ")[1]))
            t = float(data[7].split(" = ")[1])

            ct.displacement.append(d*1e6)
            ct.temp_junc.append(t)
            ct.t_metal_top.append(t_list[0]*1e6)
            ct.t_metal_bottom.append(t_list[1]*1e6)
            ct.t_metal_side.append(t_list[2]*1e6)


# Ensuring the script is only imported into other programs as intended
if __name__ == "__main__":
    print("This script is only meant to be implemented as a module, not run
independently.")
```

advanced_plotter.py

```
"""Program contaning all necessary functions to plot figures and
   somewhat automate that process. Held for other programs to access.

   Inputs:
         solver.py: Plotting functions are called from within solver.py with
any parameters necessary passed to them

   Outputs:
             solver.py: Outputs optimum values from plots to solver.py to be
printed to user
         figure_*.png: Saves plots of output data as figures in png format
"""

# Imports
import numpy as np
from matplotlib import pyplot as plt
from typing import List
from auto_sweep import *

# Function Definitions


def adj_r(x: List[float], y: List[float], degree: int) -> float:
    """Takes a set of data and tries to fit a polynomial curve to it.
       Outputs the adjusted R Squared value of the fit to see how well
       the curve fits the data.

       Args:
           x (List[float]): The x-axis data of the set to be plotted
           y (List[float]): The y-axis data of the set to be plotted
           degree (int): Degree of polynomial to try and fit to data

       Returns:
           r_squared (float): The adjusted R Squared value of the fit
                              representing how well the curve fits the data
    """

    # Fit polynomial of provided degree to data
    coeffs = np.polyfit(x, y, degree)
    p = np.poly1d(coeffs)

    # Get y hat and y bar from curve fit to x-axis data and given y-axis
    # data resepectively to see what the curve predicts should be the
    # y-values in comparison to the actual y-values
    y_hat = p(x)
    y_bar = np.sum(y)/len(y)

    # Calculate Sum of Squares Regression and Total Sum of Squares
    ssreg = np.sum((y_hat - y_bar)**2)
    sstot = np.sum((y - y_bar)**2)

    # Calculate adjusted R Squared value from previous computations
    r_squared = 1 - (((1 - (ssreg/sstot))*(len(y) - 1))/(len(y) - degree -
1))

    return r_squared


def plotter_2d(x: List[float], y: List[float], count: int, title: str,
x_label: str, y_label: str, maximize=True, max_degree=5) -> List[float]:
    """Takes a set of data and plots a 2D figure with it given chart titles,
       a figure number, and other optional parameters. Returns the
       optimum point in the data after fitting to a curve. Saves the
       resulting figure as a png image in the working directory.

       Args:
           x (List[float]): The x-axis data of the set to be plotted
           y (List[float]): The y-axis data of the set to be plotted
           count (int): The figure number to output in the filename
           title (str): The figure title
           x_label (str): The x-axis data label for the figure
           y_label (str): The y-axis data label for the figure
           maximize (bool): Default value is True; Determines whether
                            optimum value to be searched for is a minimum
                            or maximum of the given data; True yields
                            a maximum, False yields a minimum
           max_degree (int): Default value is 5; Maximum degree of
                             polynomial to try and fit to the data

       Returns:
           optimum (List[float]): The optimum datapoint that matches
                                  the requested criteria based on the curve
                                  that best fits the data
    """

    # Calculate adjusted R Squared value for data using degrees
    # of polynomials until the maximum degree specified
    r_squared = [adj_r(x, y, i)
                 for i in range(1, max_degree + 1)]
```

```
# Use the degree that best fit the data going forward
degree = r_squared.index(max(r_squared)) + 1

# Fit a model to the data with the best degree polynomial
model = np.poly1d(np.polyfit(x, y, degree))
polyline = np.linspace(min(x), max(x), 100)

# Find critical points of curve fit using first derivative test
bounds = [min(x), max(x)]
crit_x = bounds + [i for i in model.deriv().r if i.imag ==
                   0 and bounds[0] < i.real < bounds[1]]
crit_pts = zip(model(crit_x), crit_x)

# Depending on the requested optimum, find
# the maimum or minimum of the critical points
if maximize == True:
    optimum = max(crit_pts)
else:
    optimum = min(crit_pts)

# Re-organize the optimal point so it is
# in the format (x, y) and is entirely real
optimum = [i.real for i in optimum][::-1]

# Initialize figure
fig = plt.figure()

# Plot FlexPDE data as a scatter in default blue
plt.scatter(x, y, marker="o")

# Plot polynomial as a curve in red
plt.plot(polyline, model(polyline), color="red")

# Mark optimum point with green triangle
plt.scatter(optimum[0], optimum[1], color="green", marker="^")

# Add legend and chart titles
plt.legend(["FlexPDE Data", "Polynomial Fit", "Maximum"], loc='best')
plt.title(title)
plt.xlabel(x_label)
plt.ylabel(y_label)

# Set up chart grid
plt.grid(visible=True, which="major")
plt.minorticks_on()
plt.grid(visible=True, which="minor", linewidth=0.25)
```

```
    # Save figure and clear plotter for next figure
    plt.savefig(f"figure_{count}.png", format="png", dpi=500)
    fig.clear()
    plt.close(fig)

    return optimum


def plotter_3d(x: List[float], y: List[float], z: List[float], count: int,
title: str, x_label: str, y_label: str, z_label: str, maximize=True) ->
List[float]:
    """Takes a set of data and plots a 3D figure with it given chart titles,
       a figure number, and other optional parameters. Returns the
       optimum point in the data. Saves the resulting figure
       as a png image in the working directory.

       Args:
           x (List[float]): The x-axis data of the set to be plotted
           y (List[float]): The y-axis data of the set to be plotted
           z (List[float]): The z-axis data of the set to be plotted
           count (int): The figure number to output in the filename
           title (str): The figure title
           x_label (str): The x-axis data label for the figure
           y_label (str): The y-axis data label for the figure
           z_label (str): The z-axis data label for the figure
           maximize (bool): Default value is True; Determines whether
                            optimum value to be searched for is a minimum
                            or maximum of the given data; True yields
                            a maximum, False yields a minimum

       Returns:
           optimum (List[float]): The optimum datapoint that matches
                                  the requested criteria
    """

    # Initialize figure
    fig = plt.figure()
    ax = plt.axes(projection="3d")

    # Depending on the requested optimum, find
    # the maimum or minimum of the points
    if maximize == True:
        optimum = [x.pop(z.index(max(z))), y.pop(
            z.index(max(z))), z.pop(z.index(max(z)))]
    else:
        optimum = [x.pop(z.index(min(z))), y.pop(
            z.index(min(z))), z.pop(z.index(min(z)))]

    # Plot FlexPDE data as a scatter in default blue
    ax.scatter(x, y, z, marker="o")

    # Mark optimum point with green triangle
    ax.scatter(optimum[0], optimum[1], optimum[2], color="green", marker="^")

    # Add legend and chart titles
    ax.legend(["FlexPDE Data", "Maximum"], loc='best')
    ax.set_title(title)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_zlabel(z_label)

    # Set up chart grid
    ax.grid(visible=True, which="major")
    ax.minorticks_on()
    ax.grid(visible=True, which="minor", linewidth=0.25)

    # Save figure and clear plotter for next figure
    plt.savefig(f"figure_{count}.png", format="png", dpi=500)
    ax.clear()
    plt.close(fig)

    return optimum


def plotter_3d_contour(x: List[float], y: List[float], z: List[float], c:
List[float], count: int, title: str, x_label: str, y_label: str, z_label:
str, c_label: str, maximize=True) -> List[float]:
    """Takes a set of data and plots a 3D figure with it with colour
       representing a fourth dataset given chart titles, a figure number,
       and other optional parameters. Returns the optimum point in the data.
       Saves the resulting figure as a png image in the working directory.

       Args:
           x (List[float]): The x-axis data of the set to be plotted
           y (List[float]): The y-axis data of the set to be plotted
           z (List[float]): The z-axis data of the set to be plotted
           c (List[float]): The data to be plotted over the colour gradient
           count (int): The figure number to output in the filename
           title (str): The figure title
           x_label (str): The x-axis data label for the figure
           y_label (str): The y-axis data label for the figure
           z_label (str): The z-axis data label for the figure
           c_label (str): The colour gradient data label for the figure
           maximize (bool): Default value is True; Determines whether
                            optimum value to be searched for is a minimum
                            or maximum of the given data; True yields
                            a maximum, False yields a minimum

       Returns:
           optimum (List[float]): The optimum datapoint that matches
                                  the requested criteria
    """

    # Initialize figure
    fig, (ax, cax) = plt.subplots(ncols=2, figsize=(
        7.2, 4.2), gridspec_kw={"width_ratios": [1, 0.035]})
    fig.tight_layout(pad=3)
    fig.subplots_adjust(left=0, right=0.89)

    ax.remove()
    ax = fig.add_subplot(1, 1, 1, projection="3d")

    # Depending on the requested optimum, find
    # the maimum or minimum of the points
    if maximize == True:
        ind_c = c.index(max(c))
    else:
        ind_c = c.index(min(c))

    optimum = [x[ind_c], y[ind_c], z[ind_c], c[ind_c]]

    # Plot FlexPDE data as a scatter with colour varying with capacitance
    plot = ax.scatter(x, y, z, c=c, marker="o", cmap="gist_rainbow_r")

    # Add colour bar and chart titles
    fig.colorbar(plot, cax=cax, orientation="vertical",
                 extend="neither", label=c_label)

    ax.set_title(title)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_zlabel(z_label)

    # Set up chart grid
    ax.grid(visible=True, which="major")
    ax.minorticks_on()
    ax.grid(visible=True, which="minor", linewidth=0.25)
```

```
    # Save figure and clear plotter for next figure
    plt.savefig(f"figure_{count}.png", format="png", dpi=500)
    ax.clear()
    plt.close(fig)

    return optimum


# Ensuring the script is only imported into other programs as intended
if __name__ == "__main__":
    print("This script is only meant to be implemented as a module, not run
independently.")
```

solver.py

```
"""Main program to be run. Uses functions from other programs and
   solves for optimum in given problem.

   Inputs:
             constants.py: Retrieves output data from global variables
         auto_sweep.py: Uses functions to run FlexPDE scripts in a
multithreaded manner
         advanced_plotter.py: Uses functions to plot data, fit curves, and
identify optima

   Outputs:
             None
"""

# Imports
import constants as ct
from advanced_plotter import plotter_3d_contour
from auto_sweep import *

# Function Definitions


def main():
    """Main function of main program. Uses functions from other programs and
       solves for optimum in given problem.

       Args:
           None

       Returns:
           None
    """

    for (ind, material) in enumerate(ct.MATERIAL_LS):
        # Begin range at minimum possible and maximum possible thicknesses
        initialize_range([ct.MIN_THICK, ct.MAX_THICK], [
                         ct.MIN_THICK, ct.MAX_THICK], [ct.MIN_THICK, 
ct.MAX_THICK])

        # Sweep and organize data
        sweep(material)
        collate_data()

        # Initialize variables for figure plots
        fig_counter = ind + 1
        title = f"Displacement of a Piezoelectric Cantilever Varying 
{material} Thickness"
        x_label = f"Top {material} Thickness (\u03BCm)"
        y_label = f"Bottom {material} Thickness (\u03BCm)"
        z_label = f"Side {material} Thickness (\u03BCm)"
        c_label = "Tip Displacement (\u03BCm)"

        # Plot data and output maximum
        maximum = plotter_3d_contour(ct.t_metal_top, ct.t_metal_bottom, 
ct.t_metal_side,
                                     ct.displacement, fig_counter, title, 
x_label, y_label, z_label, c_label)

        # Open FlexPDE script and write formatted template
        with open("frequency.pde", "w") as stdin:
            print(ct.SECONDARY_TEMPLATE_SCRIPT.format(
                ct.E_LS[ind], ct.NU_LS[ind], maximum[0]*1e-6, maximum[1]*1e-
6, maximum[2]*1e-6, ct.ALPHA_T_LS[ind], ct.RHO_LS[ind]), file=stdin)

        # Adjust timeout according to computing power
        # (if scripts return with timeout, increase timeout)
        completed = run([ct.FLEX_VERSION, "-S", "frequency.pde"], timeout=30)

        # Output runcode of file to confirm successful completion
        print(f"File frequency returned {completed.returncode}")

        with open(f"frequency_0.txt", "r") as stdin:
            freq = float(stdin.readlines()[5].split(" = ")[1])

        print(f"--{material} Maximum--")
        print(
            f"t_metal_top: {maximum[0]} (\u03BCm), t_metal_bottom: 
{maximum[1]} (\u03BCm), t_metal_side: {maximum[2]} (\u03BCm)")
        print(
            f"displacement: {maximum[3]} (\u03BCm), temp_junc: 
{ct.temp_junc[ct.displacement.index(maximum[3])]} (\u00b0C), res_freq: {freq} 
(Hz)")


# Run main() when run as main program; Not meant to be imported as a module
if __name__ == "__main__":
    main()
```