Long-Term Stability Aging Study of Silicon Nitride Nanomechanical Resonator

MICHEL STEPHAN

Thesis submitted to the University of Ottawa
In partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE IN MECHANICAL ENGINEERING

Department of Mechanical Engineering
Faculty of Engineering
University of Ottawa

Abstract

The resonance frequency of a silicon nitride (SiN) nano-electromechanical systems (MEMS/NEMS) can be measured precisely due to their large quality factor that is associated to low thermomechanical fluctuations. While these properties enable the fabrication of high performance sensors, their use will eventually raise questions regarding their long-term stability, notably for calibration purposes. The long-term frequency stability and aging of SiN are less studied than the short-term fluctuations such as thermomechanical noise. Long-term aging studies exist for quartz clocks as well as MEMS silicon clocks and accelerometers, but not for SiN resonators with high quality factors. Thus, in this work we conduct the aging study of SiN membranes fabricated by our lab, by constantly tracking changes of the resonance frequency of the device over a long period. The evolution of the frequency drift is tracked, by optical interrogation, continuously for 135 days with a digital phase locked loop (PLL). Our device is placed in a cell under high vacuum to suppress air damping on our resonating membrane. Furthermore, due to its high sensitivity to temperature changes, our silicon nitride resonator and vacuum chamber are placed in an air bath providing a stable temperature (within 0.5 K over 135 days in the present case). To compensate further the frequency drifts induced by temperature changes, a multimeter measures the resistance of a calibrated thermistor placed inside the vacuum environment. The measured frequency drift for the aging periods of 135 days was of 300 parts per million (ppm) and was consistent with previously reported double logarithmic models for quartz oscillators. The initial stage of negative frequency drift, in our aging data, is consistent with the behaviour expected from the desorption of water due to the transition from ambient air environment to high vacuum. We review models explaining how water adsorption/desorption impacts our membrane's frequency by (1) inducing chemical reaction stresses (most important

effect), (2) through the contribution of the water surface tension stress (non-negligible effect), and (3) through mass loading from water molecules (weakest effect). After this initial negative trend, the membrane frequency drift inverts and increases almost linearly, in a fashion consistent with loss of mass from desorption of other chemical species. To identify these chemical species, X-ray photoelectron spectroscopy measurements were conducted on a reference membrane stored in an ambient setting and on our membrane placed under vacuum during our aging studies. The aged membrane, compared to its reference counterpart, contained substantially less alkaline ion contaminants (i.e., sodium, calcium and potassium), most likely due to desorption of these species during the aging measurement, and to the increase in adsorption occurring on the reference membrane concurrently. We therefore hypothesize that trapped negative charges, which is a typical phenomenon within dielectric materials such as SiN, might progressively attract positive ion contaminants over time when the device is exposed to ambient air.

Acknowledgments

This work is ultimately the result of a contribution between many. I have gained an imaginable amount of engineering, physic, and cultural knowledge in a small span of two years.

أودّ تكريس هذا العمل لأخي كريستوف وأعمامي وخالاتي واولادهم. لقد كنتم جميعًا مصدرًا هائلاً للتشجيع والدعم. كنتم دائمًا تثقون فيَّ كثيراً وتؤمنون دائمًا بقدراتي على الرغم من أي موقف. لقد دفعني حبكم الثمين للتغلب على كل العقبات وعلى إنتاج هذه الأطروحة التي أنا فخور بها جدًا. إلى والدي رولان وأمي بيتينا ، أجدادي ميشال وجورجيت وجدي المرحوم فؤاد ، أهدي هذا العمل لكم جميعًا. أنّني فَخور بعملي الرائع ، إلّا أنه لا يقترب من أيٍ من إنجازاتكم ألعظيمة التي سمحت لي بحياة جميلة وناجِحة ، على أقل تقدير. أشكركم من عمق قلبي على دعمكم المستمر ، ويشرّفني للغاية أن أجعلكم جميعًا فَخورين.

Pou pi bon mwatye mwen an, mèsi deske ou te toujou bò kote m, menm lè mwen te anba anpil presyon. Ou te toujou rasire mwen e demontre m, ak pèseverans e detèminasyon w, ke malgre eprèv lavi, yon moun ka toujou reyalize rev li yo. Sajès ou, lanmou w, ankourajman w ak anpil lòt bèl kalite w te pèmèt mwen reyalize tèz ekstraòdinè sa. Mwen enfiniman rekonesan anvè ou.

À mon superviseur, professeur Raphael St-Gelais, je désire exprimer mon immensurable appréciation. Tu es un splendide mentor et source de support, tout étant une personne que tous aiment côtoyer. Tu m’as incessamment motivé, ainsi que mes pairs, afin qu’on puisse se surpasser tout en repoussant continuellement les limites de nos connaissances. Je ne peux te remercier suffisamment pour l’opportunité que tu m’as dédiée et pour les innombrables conseils que tu m’as transmis au cours de ma thèse.

My dear colleagues at UOMEMS lab at the University of Ottawa, notably Alexandre Bouchard, Timothy Hodges, Mathieu Giroux, Chang Zhang and Gengyang Mu, without all of you I would have never found the drive and inspiration to complete my thesis. I want to thank you for

the continuous support and fun we had together. I have learned so much on many different facets by simply having you around me and interacting as friends.

Dr. Richard Green, Dr. Triantafillos Koukoulas and Dr. Lixue Wu from the Metrology Research Council of the National Research Council of Canada, your special expertise and guidance have been crucial to my success and understanding of the phenomena studied. I wish to convey my deepest gratitude for the time and efforts you have attributed to me.

I would also like to thank the machine shop at the University of Ottawa for their quick and amazing work. Having access to a machine shop locally and without any fees greatly accelerates our projects while avoiding many complications, an opportunity that many do not have!

Table of Contents

List of Figures

List of Tables

Nomenclature

List of Abbreviations

APC	Angle Polished Connector
CAD	Computer Aided Design
CMOS	Complementary Metal-Oxide Semiconductor
DFT	Discrete Fourier Transform
DUT	Device Under Test
FFT	Fast Fourier Transform
GUI	Graphical User Interface
HVAC	Heating, Ventilation and Air Conditioning
LAN	Local Area Network
LPCVD	Low Pressure Chemical Vapour Deposition
MCXO	Microcomputer Compensation Crystal Oscillator
MEMS	Nano-Electromechanical Systems
NEMS	Micro-Electromechanical Systems
NRC	National Research Council Canada
OCXO	Oven Controlled Crystal Oscillator
PC	Polished Connector
PLL	Phase-Locked Loop
ppm	Parts per Million
RTD	Resistance Temperature Detector
SiN	Silicon Nitride
SPXO	Simple Packaged Crystal Oscillators
TCE	Thermal Coefficient of Elasticity
TCF	Thermal Coefficient of Frequency
TCXO	Temperature Compensation Crystal Oscillator
XPS	X-Ray Photoelectron Spectroscopy
ZnSe	Zinc Selenide

List of Latin Letters

A_m	Are of the silicon nitride membrane	m^2
A_{sites}	Area of the silicon nitride active sites	m^2
a	First coefficient of the RTD	-
b	Second coefficient of the RTD	-
d	Distance between the fibre tip and the moving membrane	nm
E_b	Desorption energy barrier	J
E_s	Modulus of elasticity of the substrate	MPa
f	Frequency of the resonator	Hz
f_{occ}	Average site occupation	-
f_{Temp}	Temperature dependent frequency of our membrane	Hz
f_{Age}	Frequency drift due to aging of our membrane	ppm
h	Thickness of the resonator	mm
k_B	Boltzmann constant	J/K
k_{eff}	Effective stiffness of the resonator	N/m
L	Dimension of the square membrane	mm
L_x	Dimension of the rectangular membrane along the X axis	mm
L_y	Dimension of the rectangular membrane along the Y axis	mm
M_c	Molar mass of contaminants	kg/mol
M_{SiN}	Molar mass of silicon nitride	kg/mol
m	First integer of the vibration mode of the resonating membrane	-
m_{cont}	Mass of a molecule of adsorbate	kg
m_{eff}	Effective mass of the resonator	kg
m_m	Mass of the membrane	kg
N_A	Avogadro's number	1/mol
N_c	Number of contaminant molecules in a monolayer of our device	-
n	Second integer of the vibration mode of the resonating membrane	-
P	Partial pressure of the contaminant compared to its environment	Pa
P_w	Partial vapour pressure of water	Pa

P_{ws}	Saturation partial vapour pressure of water	Pa
Q	Quality factor	-
$Q_{clamping}$	Quality factor limited by clamping dissipations on the resonator	-
$Q_{intrinsic}$	Quality factor limited by intrinsic dissipations of the resonator	-
Q_{medium}	Quality factor limited by losses in the medium of the resonator	-
Q_{other}	Quality factor limited by losses due to minor dissipation mechanisms	-
r_a	Contaminant adsorption rate	Hz
S_A	Surface density of adsorption site	sites/m^2
S_V	Volumetric density of adsorption site	sites/m^3
s	Sticking coefficient between the adsorbates and the adsorbent	-
T	Temperature of the environment where adsorption/desorption occurs	K
T_{Cell}	Temperature of the vacuum cell in the controlled environment	K
T_{Lab}	Temperature of the ambient environment of the lab	°C
T_{Ref}	Reference temperature for the thermistor	K
t	Thickness of our membrane	m
t_{Age}	Time of aging of our membrane	S
V	Volume of our silicon nitride membrane	m^3
V_{max}	Maximum voltage due to constructive interference in photodetector	V
V_{min}	Minimum voltage due to destructive interference in photodetector	V
V_{out}	Photodetector output voltage	V

List of Greek Letters

β	Temperature coefficient of the thermistor	K
Δf	Frequency change of the resonator	Hz
Δm_c	Total mass change due to adsorbates	kg
$\Delta\sigma_{ST}$	Additive surface tensile stress	MPa
δs	Surface tension	N/m
λ	Wavelength of the laser	nm
ν_d	Desorption attempt frequency	Hz
ρ	Density of the membrane	kg/m^3
ρ_r	Density of the resonator	kg/m^3
ρ_s	Density of the substrate	kg/m^3
ρ_{SiN}	Density of silicon nitride	kg/m^3
σ	Stress of the resonating membrane	MPa
$\sigma_{Initial}$	Initial stress of our membranes	MPa
τ	Vibration decay rate obtained from exponential curve-fit	s
φ	Environmental relative humidity	%
$\Omega_{m,n}$	Resonance angular eigenfrequencies	rad/s
Ω_N	Nominal resistance of the RTD	Ω
Ω_{Ref}	Reference resistance of the thermistor	Ω
Ω_{RTD}	Measured resistance of the RTD	Ω
Ω_{Th}	Measured resistance of the thermistor	Ω

1. Introduction

1.1 Context

In today's highly technological era, nano, and micro-electromechanical systems (NEMS and MEMS) are, without most of us realizing, some of the most common and versatile instruments in our everyday life. The tire pressure monitoring system used in most modern vehicles to detect flat tires, the accelerometers in handheld devices that sense the orientation of the device and position the screen accordingly, even the temperature sensors used in computers or other electronic systems are few examples of MEMS. These small form factor devices rely on their mechanical and electrical systems to achieve desired functions or measurements.

The mass fabrication of these multifaceted devices makes their production and usage exceedingly desirable in today's highly commercially oriented world. Their compact format combined with the fact that they can be fabricated from different materials makes them easy to implement in many different applications such as the human body [1], mobile phones [2], tide monitoring [3] and much more.

Vibrating devices, also called resonators, are an important subset of MEMS devices. Resonators vibrate at a particular natural frequency that depends on numerous factors such as the mechanical properties of the material from which they are made, the physical dimensions of the MEMS as well as the environment in which they vibrate. The quality factor (*Q-factor)* of a vibrating device is a characteristic used to quantify the level at which a resonator is underdamped, an important property to ensure high performances for the devices.

The frequency of resonating devices shifts as they are affected by changes to their environment or their structure. Analyzing the shift of frequency is thus used to quantify various occurring physical phenomena, whether it be an applied force, a change in pressure or a change of temperature. MEMS resonators can also be used in silicon clocks for timing applications in various fields [4]. Their frequency can be designed in a wide range, typically 1 - 125 MHz, making them more versatile than classic quartz resonators used in clocks. In MEMS gyroscopes, disk resonators are excited in resonance in order to become sensitive to angular motion via the Coriolis effect [5]. Silicon nitride (SiN or Si_3N_4) based MEMS resonators, which are mostly at the research phase, are used by our lab [6] and others [7], as heat radiation sensing devices due to their ability to achieve high sensitivity to temperature and incident radiation. Once exposed to a source of radiation, heat is transferred to the MEMS, which will relax its internal tensile stresses in reason of the thermal expansion its materials. The quantification of the change of temperature due to radiation is therefore done by measuring the natural frequency variations of the device that are induced by these stress fluctuations. [6].

Shift of resonance frequency in MEMS/NEMS can be caused by (1) a quantifiable occurring physical phenomenon that we want to measure, or (2) unwanted instabilities that we wish to suppress as much as possible. Sources of instabilities that will affect the performance of MEMS can be classified as short-term effects, or long-term effects that cause drift over the course of many weeks. Thermomechanical fluctuations cause instant short-term instabilities. These fluctuations were studied intensively for silicon nitride nanomechanical resonators [8]–[11]. Essentially, due to thermal energy, the atoms within the materials are constantly moving and colliding between each other. These excitations can couple to the resonator as a whole, creating instabilities and frequency fluctuations [12].

While these short-term thermomechanical effects are widely understood, instabilities from long-term effects have not been as widely studied in the context of the emerging field of silicon nitride resonators. Long-term instabilities are expected to be caused by the aging of the device's materials, or by slow adsorption/desorption of contaminants, which we aim to quantify in this work.

1.2 Problem Statement and Objectives

Aging and frequency drift have widely been studied in silicon devices used as clocks [4], but these phenomena have yet to be researched in the emerging field of silicon nitride resonators, which is an important material platform for many studies performed by our lab and others. Thus, our goal is to fill the gap of missing knowledge when it comes to the different instabilities affecting SiN resonators. This is achieved by studying the frequency drift caused by aging, which was much less studied than short-term instabilities such as thermomechanical fluctuations [8].

Aging and drift are often ambiguously used or even confused for one another, thus clear boundaries and definitions are required. The *International Telecommunication Union*'s [13] definition for aging is: "The systematic change in frequency with time due to internal changes in the oscillator. NOTE 1 – It is the frequency change with time when factors external to the oscillator (environment, power, supply, etc.) are kept constant." On the other hand, the organization defined drift as follows: "A systematic undesired change in the frequency of an oscillator over time. Drift is due to aging plus changes in the environment and other factors external to the oscillator."

Our goal for this work is to quantify the aging of our SiN resonating membranes, which is achieved by measuring the occurring frequency drift. Specific measures are implemented to suppress, as much as possible, all instability sources causing drift other than aging.

While the study of the frequency drift in silicon nitride resonators is interesting on its own, we also have a specific objective that stems from our collaboration with metrologists at the National Research Council of Canada (NRC). This collaboration notably aims at developing an optically interrogated accelerometer that must meet certain requirements in order to be qualified for use in the field of metrology. An important performance criterion of this device is that it is required to be stable over a long period of time. As stated by our collaborator, a target of 1 ppm (parts per million) of frequency drift over a period of a year would be an ideal stability criterion, allowing SiN devices to be used in new applications for the NRC.

In summary, our objectives are:

- Develop an experimental platform to study the long-term frequency stability of SiN resonators,
- Measure the long-term stability of SiN resonators fabricated by our lab.

1.3 Silicon Nitride Nanomechanical Sensors

While it shares common advantages of other materials used in the field of MEMS, SiN is mainly researched due to its ability to create devices achieving large mechanical quality factors (*Q-factor* or *Q*), which in turn allow high precision measurements. This characteristic can be defined as the ratio of the energy that is stored in a vibrational mode to the energy dissipated during a single oscillation [12]. An oscillating device loses its energy due to different damping mechanisms caused by conditions external to the device, such as friction with air in the ambient environment, or by internal losses that depend on the material's modulus of elasticity.

The higher the quality factor, the more the device retains its energy whilst oscillating over time. The device can thus provide a more stable short-term frequency readout leading to higher levels of precision for measurements.

Damping dilution is a phenomenon that effectively enhances the quality factor of a device. When a tensile stress source is applied, materials are subjected to elongations, thus allowing more energy to be stored in the resonator. On the other hand, introducing tensile stress does not affect the rate of energy dissipation whilst the resonator vibrates [12]. Applying such stress source to a vibrating device will dilute the occurring energy damping and consequently result in a larger Q as per its established definition.

SiN's mechanical properties enables it to be preloaded with considerable amount of tensile stress without yielding, making it an enviable material to produce resonators with high *Q-factors.* This property withheld by SiN is why it has recently been favoured to fabricate diverse MEMS resonators used for acceleration [14], pressure [15] and even temperature sensing [16].

The high quality-factor achievable with SiN provides a better short-term frequency stability, by isolating the device from thermomechanical fluctuations. This link between low dissipation and high stability is clearly established by the fluctuation dissipation theorem [17]. On the other hand, the scale of the effects of aging on the drift sustained by a SiN MEMS over a long period of time is unknown. Since our SiN resonators are highly sensitive to temperature changes, we have taken specific measures to reduce the effects of frequency variations induced by temperature fluctuations. This ensures that the long-term stability measurements reflect, as much as possible, the effects of aging on the performance of the membrane.

2. Background and Literature Review

2.1 Background on Silicon Nitride Nanomechanical Resonators

Recently, resonator devices have been manufactured from SiN, since it offers the ability to achieve large *Q-factor*s, which greatly decouples them from unwanted environmentally induced changes [18] meaning their measurements are more representative of the phenomena of interest.

The resonance angular eigenfrequencies (in rad/s) of a rectangular membrane ($\Omega_{m,n}$) are given by:

$$\Omega_{m,n} = \sqrt{\frac{\sigma}{\rho}}\sqrt{\frac{m^2\pi^2}{L_x^2} + \frac{n^2\pi^2}{L_y^2}}, \tag{1}$$

where σ is the stress of the membrane, ρ is its density, m and n are integers representing the vibrational eigenmode order, and L_x as well as L_y are its dimensions. In this work, our membrane, which is sketched in the **Figure 1**, has a square geometry where $L_x = L_y = 6$ mm.

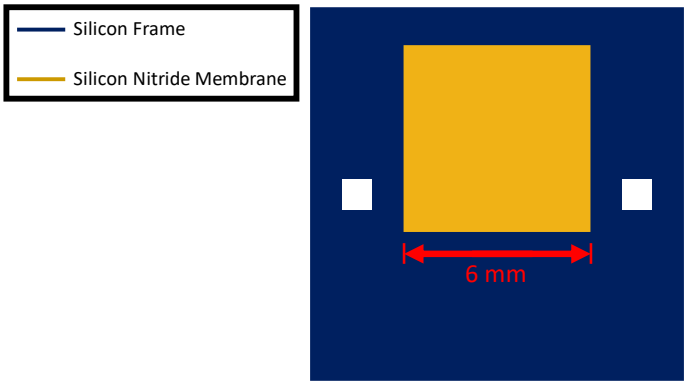


Figure 1: Simplified schematics of our SiN membranes.

The *Q-factor* for SiN resonators is well studied and can reach high values above 10^7 at extremely low temperature [18], but is mainly limited by material dissipation.

2.2 Short-term Dissipation Mechanisms in SiN Resonators

Maximizing the *Q-factor* of our resonators is essential to suppress short-term or sporadic instability sources, however a high *Q* device might still suffer from poor long-term stability, which is the focus of this work. This section is therefore presented for completeness of the basis of SiN resonators, but is not central to our work on aging and is far more documented in Schmid's work [12]. A reader with limited time might want to give more attention to section 2.1 and 2.3 to 2.5.

The quality factor of a resonator is mainly dictated by the sum of its different types of dissipation mechanisms. Schmid largely elaborates on the most important mechanisms, which are dissipations due to the medium of the device, its clamping as well as intrinsic dissipations inside the device [12] (e.g., internal friction). All additional sources of dissipations do not individually have a large enough magnitude to be considered separately, they are thus grouped and labelled as other dissipations [12]. The following equation is used to calculate the theoretical *Q-factor* of our MEMS [12]:

$$\frac{1}{Q} = \frac{1}{Q_{medium}} + \frac{1}{Q_{clamping}} + \frac{1}{Q_{intrinsic}} + \frac{1}{Q_{other}}. \quad (2)$$

2.2.1 Medium Dissipations

The magnitude of dissipation for a resonating MEMS is largely affected by the type of medium in which it is located. The different media can be classified from the most to least dissipating: viscous liquid, viscous fluidic gas, and rarefied ballistic gas [12]. When the resonator is submerged in a liquid the medium is considered as a viscous liquid. This type of medium, especially due to its high density, can significantly dampen a resonator and potentially overdamp it, which would prevent it from effectively vibrating and producing resonance [12]. The next two media, which are comprised of gas, have squeeze-film and drag-force damping loss mechanisms.

When the representative dimension of the MEMS is larger than the mean free path of the gas, the medium is viscous fluidic. The importance of the aforementioned loss mechanisms will depend on the geometrical parameters, density and angular velocity of the resonator's oscillation as well as the viscosity of the gas. Additionally, squeeze-film damping also depends on the distance between the resonating material and the closest wall [19]. Comparing squeeze-film and drag-force models reveals that they have an equal dampening contribution when the width of the device is twice as big as the distance between the resonating material and a fix boundary. As the width grows larger than the value necessary to achieve that equilibrium ratio, the squeeze-film dampening mechanism becomes more important [12]. Increasing the mean free path of gas, notably by decreasing the pressure of the environment, to a point where it is larger than the representative length of the device makes the medium a rarefied ballistic gas [12]. In this environment, the collisions occurring between the gas and the oscillating MEMS dictate the magnitude of dissipation. The pressure level in the environment will linearly affect the magnitude of the new squeeze-film and drag-force dampening. Temperature, molar mass and the universal molar gas constant of the media are new parameters that will affect these dissipation mechanisms [12].

In the context of this work, the vacuum system implemented and explained in section 3.2.1, allows the device's environment to reach and maintain, through time, pressures on the scale of 10^{-8} Torr. This ensures that the medium in which our MEMS oscillates is a rarified ballistic gas allowing for a high quality-factor.

2.2.2 Clamping Dissipations

Anchoring a resonator will couple its vibrating body with other solid surfaces or media, including its own substrate, to which a part of its vibrational energy will be transferred and lost [12]. The geometry and properties of the resonator, compared to its supporting frame, will have a

large impact on the clamping restrictions that limit the *Q-factors*. Schmid developed models to compute the theoretical clamping quality factor ($Q_{clamping}$) limitation between a membrane and its substrate, which is presented in the following equation [12]:

$$Q_{clamping} \approx 1.5 \sqrt{\frac{\rho_r}{\rho_s}} \frac{E_s^{\frac{3}{2}}}{\sigma^{\frac{3}{2}}} \frac{n^2 m^2}{(n^2+m^2)^{\frac{3}{2}}} \frac{L}{h}, \tag{3}$$

where ρ_r and ρ_s are respectively the density of the resonator and the substrate in kg/m^3, E_s is the modulus of elasticity of the substrate in MPa, n and m are the vibrational modes of the resonator, σ is its stress in MPa, L is the side length of the membrane in m as well as its thickness h in m.

Thus, to minimize energy losses between a resonating membrane and its substrate, it is necessary to achieve a large aspect ratio between its side length and thickness. Furthermore, ensuring that the substrate has a large modulus of elasticity while also having a lower density than the resonator restricts the potential vibration losses. Equation (3) also showcases how the clamping losses are more important at lower modes due to their smaller vibrational energy.

Additionally, the interfacing of the substrate with other retaining components will further contribute to the clamping losses. Minimizing the contact area and force between the substrate and mounting mechanisms will impede the vibration energy transfer thus ensuring lower losses. The membrane clamping mechanism was carefully designed to maximize the achievable *Q-factor* while ensuring ease of installation of the membrane.

2.2.3 Intrinsic Dissipations

The intrinsic dissipation relates to vibration losses occurring inside the structure of the resonator. Schmid differentiates the intrinsic damping on unstressed and stressed devices. In unstressed devices, friction losses and fundamental losses are the mechanisms which are

responsible for the energy dissipation. Friction losses occur due to the interatomic clashes initiated by an exploited material defect [12]. Fundamental losses, on the other hand, are caused by entropy leading to irreversible energy dissipation as heat [12]. In stressed resonators, internal energy is mostly stored by tensile and bending stresses and is also mostly lost through bending stresses. Thus, the ratio between the tensile and bending stress of the resonator, which can also be explained through damping dilution presented in section 1.3, constructs the intrinsic dissipation in stressed resonators.

2.3 Review of Aging in Quartz Oscillator Devices

While frequency variations and *Q-factors* are well studied and deterministic for resonators made from SiN, we don't have information on the aging of these devices. We therefore review aging in other types of resonators, i.e., Quartz and Silicon.

Due to their remarkably low drift and aging over a large period of time, quartz based oscillators have been amongst standard devices used in timing applications since 1940 [20]. These devices showcase minimal aging effects leading to very small frequency drift of under a part per million (ppm) per year [21].

Quartz mineral is one of the few materials known to possess a thermal coefficient of elasticity (TCE, changes of the Young's modulus of a material as a function of temperature) that is null at room temperature while also being able to achieve, through a standard AT-cut (a planar cut of 35°15' from the XZ plane of the quartz crystal as shown in **Figure 2**), null first and second order thermal coefficients of frequencies (TCF) [20]. These properties ensure that quartz oscillators, paired with designed electronic circuits, can have a low sensibility to thermal variations

such as thermomechanical noise. This makes it much more stable through time, since it has one less source of external factors that may cause frequency drifts as time passes.

Aging of the quartz oscillator occurs on the crystal, while electronic aging affects the circuit paired with the crystal.

Quartz aging is principally due to chemisorption of contaminants, changes in its mounting stress as well as the evolution of natural imperfections in its structure [21]. Adsorption and desorption mechanisms lead to aging of the resonator, which will cause drift in a positive or negative fashion, depending on the chemical compounds in its environment. Outgassing of the crystal itself will also play a role in these contamination mechanisms and lead to aging [21]. The methods used to mount the device will introduce a source of stress due to the exerted retention forces. This clamping stress will impact the performance of the quartz oscillator. Additionally, bonding agents and clamping mechanisms are subjected to fatigue and creep, which will lead to a relaxation of the clamping stress through time [21]. Imperfections in crystals such as dislocations, presence of impurities or inclusions, lapping leading to microcracks as well as surface and point defects are a few examples of structural variations [21]. Finally, diffusion of impurities between the crystal and its electrodes as well as between the crystal and the environment is another aging mechanism with a variable rate. Depending on the materials diffused, their concentration gradient as well as other previous imperfections exploited by this mechanism, the aging it causes can have a different but significant importance [21]. The addition of all these major variations will contribute to aging of the resonator, alongside other minor sources that have not been investigated [21].

When it comes to the electronic circuit of the oscillator, the various components such as inductors, capacitors, circuit boards, resistances and diodes can age due to different factors linked

to the experimental environment such as humidity, pressure, and temperature changes, which affect the performance of the circuit and its frequency [21]. Additionally, frequency changes of the oscillator circuit can be caused by stray reactance variations originating from the displacement of electrical leads due to degradation of the circuit board [21]. Stabilizing the humidity and pressure through the use of a vacuum environment is an effective solution that drastically suppresses some of the above-mentioned circuit aging factors, thus eliminating some frequency drift [21]. While the circuit itself is affected by aging, it is independent of the resonator and won't directly contribute to the aging of the latter, but rather add drift.

With their attractive intrinsic material properties, quartz oscillators have been achieving great drift stability for many decades. Quartz resonators can feature different compensations and crystalline cuts, as illustrated in **Figure 2**, which enhance their stability through time. These oscillators can be categorized in different types from least to most stable: simple packaged crystal oscillator (SPXO), temperature compensation crystal oscillator (TCXO), microprocessor compensation crystal oscillator (MCXO) as well as oven controlled crystal oscillator (OCXO). SPXOs feature no compensation mechanisms, which makes them the most affordable oscillators and explains why they are also the most used amongst all types. They are present in most of consumer grade electronics ranging from computers to appliances which require timing circuitry and they can achieve a yearly stability of 5 to 10 ppm [21]. TCXOs and MCXOs feature temperature compensation and can achieve yearly stabilities of respectively 0.5 and 0.02 ppm [22]. Finally, OCXOs are the most stable type since they use an oven to control their operating temperature. They are said to achieve a yearly stability of 0.005 ppm for SC (stress compensated) cut crystals (a double-planar cut that features the AT cut of 35°15' from the XZ plane paired with

an additional 21°54’ cut from the YZ plane of the quartz crystal, as shown in **Figure 2** [23]), which is the optimal cut to achieve the best aging stability [22].

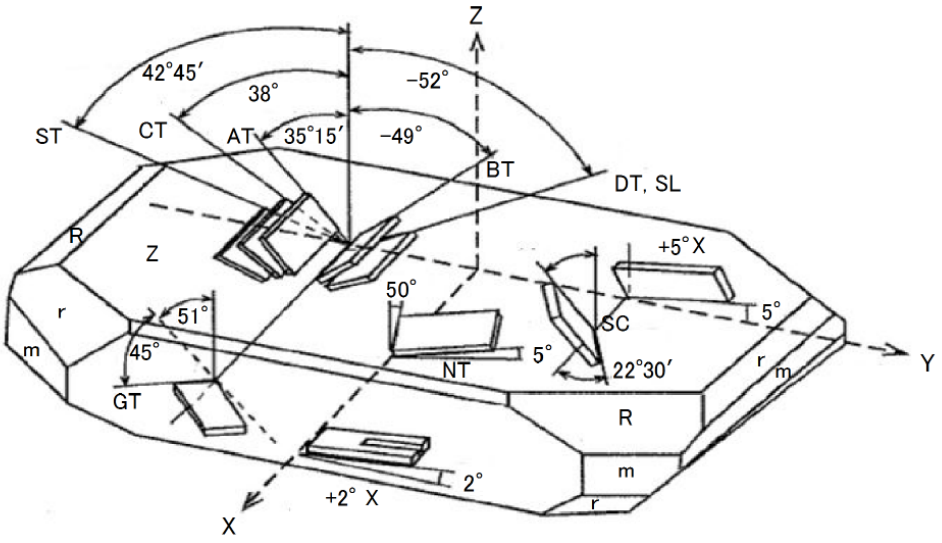


Figure 2: Quartz crystal cut orientation. Taken from [24].

2.3.1 Aging Models in Quartz Resonators

Vig [21], Landsberg [25], as well as others, have observed that aging, which is affected by a multitude of different mechanisms, can accurately be represented by logarithmic functions of time. Many researchers have observed a reversal of the aging direction of their devices, thus leading to a positive and negative frequency drift regime. A single logarithmic function could not explain this behaviour, but authors have observed that a double logarithmic function, in most cases, represent their device aging more accurately. Since aging involves multiple mechanisms as we highlighted in this section, the authors are not able to identify precisely what causes this reversal. It is hypothesized that concurrent aging mechanisms, with different rates are responsible for this phenomenon. An initial aging mechanism is responsible for the high early drift observed, but it will start to lose importance over another mechanism with an opposite behaviour, hence the reversal. The sum of both mechanisms will contribute to the logarithmic long-term aging of the device as demonstrated in the aging model presented below in **Figure 3**.

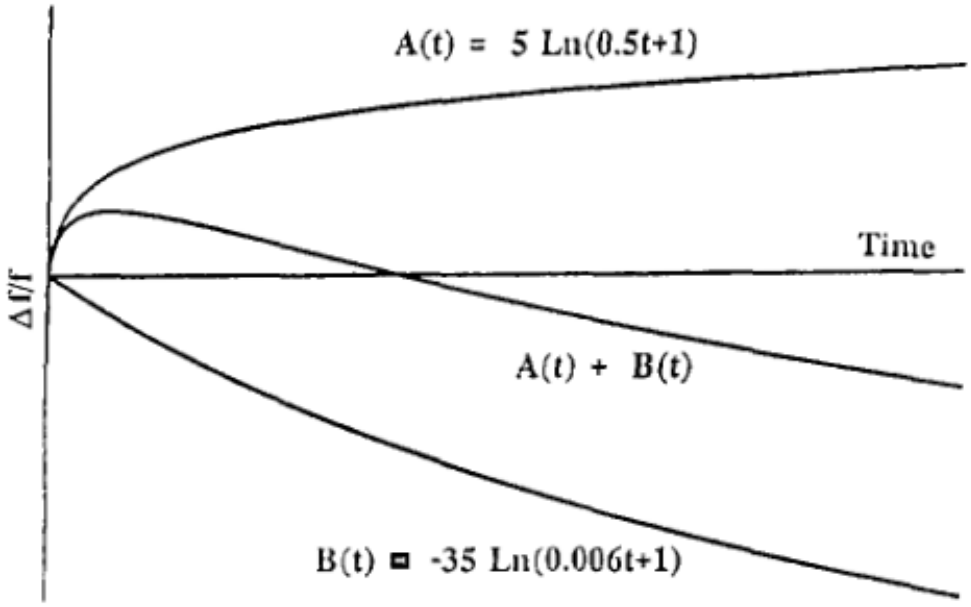


Figure 3: Typical aging behaviours. Taken from [21].

Literature [4], [21], [26]–[29] indicates that main aging mechanisms feature contamination by adsorption, mounting stress changes as well as changes in the material's properties or structure.

While contamination as well as mounting variations are observable in quartz resonators and MEMS devices, structural changes have been extensively reported for the crystal, but structural changes in silicon nitride thin films have not been studied over a long period.

2.4 Review of Aging in Silicon Oscillator Devices

Quartz crystal was the default material used to produce high-end oscillators, but in the past decades, the semiconductor industry pushed to develop chip-integrated MEMS silicon-based clocks with on par performance to quartz oscillators. Unlike standalone quartz pieces, MEMS resonators are naturally integrated on silicon chips, and are therefore much easier to integrate in semiconductor devices, resulting in compact form factor and less expensive production.

These newer and more cost-effective silicon timing devices, just as the quartz oscillators, suffer from aging notably induced by the adsorption/desorption of contaminants and stress variations in their mounting methods [4].

Silicon-based oscillators have a thermal coefficient of elasticity (TCE) of significant importance, regardless of their crystal orientation, which results in resonators with frequency variations that are highly dependent of temperature changes, as indicated by their thermal coefficient of frequency (TCF) of -28 to -30 ppm/°C [20]. Contrary to their quartz counterparts, these MEMS will be largely affected by thermal changes in their environments, which will contribute to their drift and mask the true magnitude of aging.

Thus, an important step when creating oscillators with silicon is to add measures to suppress the device's temperature sensitivity. *SiTime*, a company that develops silicon based timing devices, pairs a complementary metal-oxide semiconductor (CMOS) temperature sensor alongside their MEMS device to measure the change of temperature and compensate for it electronically [4].

The company also developed a special manufacturing and packaging process to ensure time stability performances compatible with oven compensated quartz oscillators. Since these devices are aimed to be used in timing applications, minimizing their drift, and aging is essential for the success of these MEMS clock. *SiTime* relies on its developed and well documented sealing process (*EpiSeal*^{TM}), part of their specific manufacturing process (*MEMS First*^{TM}), to ensure the lowest aging possible for their devices [30]. *EpiSeal*, which is conducted at high temperature, aims to clean the resonator's environment and seal it under high vacuum. This process begins by growing silicon layers with fine vents to form a thin envelope around the device [30]. Hot hydrogen and chlorine gas is introduced into the resonator's environment, through the vents, to effectively

cleanse it from contaminants while also ensuring all small intricate geometric features from the manufacturing process are also reached [30]. A gas mixture containing polysilicon deposits is circulated following these cleansing agents to quickly seal the vents and create the internal vacuum environment of the resonator [30]. Finally, a thick polysilicon shell is grown on the MEMS to create a robust packaging and ensure that the cleansed internal vacuum environment remains sealed through time and different operating conditions [30].

As a result of this proprietary manufacturing and packaging process, silicon MEMS created by *SiTime* have been able to reach aging drift levels of 0.02 ppm/year, which is comparable to most top end quartz oscillators devices [4].

While the aging achievements of silicon clocks are impressive, it is also relevant to study aging in other types of silicon MEMS that are used for applications other than timing. Łuczak's work reports the performance of two types of silicon-based MEMS accelerometer that were studied for periods of 4 and 10 years. While this work did not thoroughly investigate the different possible aging causes in their devices, they have reported changes in performance of their MEMS over time. The variation of the offset (in V), which is the voltage output of the device when it is static and decoupled from gravity as well as the scale factor, also commonly known as the sensitivity (in V/g), which is the voltage output per unit of acceleration of the accelerometers, are the parameters used to quantify impact of aging on the device [28]. The first type of accelerometer, which was tested over 10 years, resulted in yearly variation of 410 ppm and 678 ppm for the offset and scale factors respectively [28]. The accelerometers tested over 4 years resulted in yearly variation of 411 ppm and 1087 ppm for the offset and scale factors respectively [28].

2.5 Aging Model for SiN Resonator

In this section we describe the possible effect of the aging mechanisms identified above (section 2.3, 2.4), when applied to SiN resonator devices presented in section 2.1, for which aging models do not exist.

As reviewed above, studies on quartz and silicon-based resonators have both shown that the adsorption and desorption of contaminants will largely contribute to the aging of the device. These phenomena, which both have their own sub-mechanisms, can easily be confused with one another. Thus, a quick overview of the nomenclature is necessary. Adsorption is a phenomenon where the molecules of a contaminating fluid, labelled as adsorbates, will adhere to the surface of a solid, labelled as adsorbent. The opposite mechanism, where the adsorbates would separate from the adsorbent, is called desorption [31]. It is important to know that these phenomena are restricted to the surface of the adsorbent. Absorption occurs when adsorbates diffuse inside the bulk of the material [31]. There are two types of adsorptions: physisorption and chemisorption. When adsorbates adhere to a surface through Van der Waals interactions, physisorption is involved, while chemisorption is involved when the adsorbates form covalent or ionic bonds with the adsorbent [31].

Since no environment is ideal, even under very high vacuum, contaminants will always be present and subjected to adsorption or desorption on a resonator's surface. This type of aging will directly impact the resonance frequency of the device and cause it to drift.

In the context of silicon nitride devices, adsorption of water molecules on silicon nitride has not been studied in the context of aging, but studies exist in smaller timescales by varying humidity in the cantilever's environment and analyzing its deflection [32]. The work showed that

the mass difference due to the adsorption of water produces a very small change of the cantilever properties compared to the changes induced by the chemo-mechanical stress from the chemical reactions occurring on the resonator's surface [32].

With the various literature review conducted on different types of timing and sensing devices, we have identified that adsorption aging will cause a drift of frequency through a mass differential (described in section 2.5.2), a surface tension variation (2.5.3) as well as the inclusion of chemical stresses (2.5.4).

To simplify the models for different drifts, we consider the following case study:

- Water is assumed to be the dominating contaminant,
- The membrane's surface is covered by up to one monolayer of adsorbed water molecules.

It is likely that other contaminants are present, but the effect of water on SiN is already well documented [32] and thus easier for us to model. Likewise, conclusion drawn for water adsorption are transferrable to other potential contaminants, with parameter changes.

2.5.1 Adsorption Desorption Rates

Since our device is placed under high vacuum, the number of occupied active sites for our material will greatly vary from an ambient pressure setting. The vacuum will heavily suppress the rate of adsorption of our membrane and lead to a low density of occupied active sites, which must be found and incorporated to the different drift sources computed.

The adsorption and desorption rates as well as the fraction of occupied active sites has been computed in **Table 1** for the partial pressure of water at ambient conditions and in vacuum. Our vacuum partial pressure of contaminants is estimated at $P = 10^{-5}$ Pa, i.e., the typical pressure

achieved in our vacuum system. We assume that water is the main gas present under high vacuum, since it is a know fact that hydrophilic surfaces are a major limiting factor for the ultimate pressure of vacuum systems.

To find the ambient partial pressure of water, we use environmental parameters representative of our lab environment. Thus, the ambient pressure is of $P_{air} = 101325$ Pa, the temperature is of $T = 21°C$ and the relative humidity is approximately 50%. We then find the saturation partial vapour pressure of water (P_{ws} = 2500 Pa) at our respective dry bulb temperature using a database [33] and compute the partial vapour pressure of water (P_w in Pa) with [33]:

$$P_w = {}^{\varphi}/_{100} \cdot P_{ws}, \tag{4}$$

where φ (in %) is our environmental relative humidity. The resulting ambient partial pressure is 1250 Pa.

To find our estimated site occupancy in vacuum, we began by computing the rate of adsorption and desorption as formulated in the work of Cleland and Roukes [9]. The rate of adsorption is given by:

$$r_a = \frac{2}{5} \cdot \frac{P}{\sqrt{m_{cont} \cdot k_B \cdot T}} \cdot s \cdot A_{sites}, \tag{5}$$

where r_a (in Hz) is the adsorption rate of contaminants, which depends on the partial pressure of the contaminant compared to its environment (P in Pa), m_{cont} (in kg) is the mass of a molecule of adsorbate, k_B (in J/K) is the Boltzmann constant, T (in K) is the temperature of the environment, s is the sticking coefficient of specific adsorbates on a given adsorbent and area of the active sites. We use our experimental values for the pressure and temperature of the environment. The mass of a molecule of water ($m_{cont} = 2.991 \times 10^{-26}$ kg) is calculated in section 2.5.2, while the sticking

coefficient ($s = 0.1$) was taken from Cleland and Roukes' work [9]. The area of the silicon nitride active sites ($A_{sites} = 10^{-19}$ m^2) was taken from Stephan's work [32].

The rate of desorption is computed using:

$$r_d = \nu_d \cdot e^{\left(-\frac{E_b}{k_B \cdot T}\right)}. \quad (6)$$

In equation (6), r_d (in Hz) is the desorption rate and depends on the desorption attempt frequency (ν_d in Hz), the desorption energy barrier (E_b in J), the Boltzmann constant (k_B in J/K) and the temperature of the environment (T in K). The desorption attempt frequency ($\nu_d = 10^{13}$ Hz) is taken from Cleland and Roukes [9]. The desorption energy barrier depends on the bonds between the adsorbates and adsorbent. The energy barrier is taken from Stephan's [32] work ($E_b = 1.13 \times 10^{-19}$ J)

With the adsorption and desorption rates found, we can compute the average occupation probability of an adsorption site (f_{occ}) with the following equation [9]:

$$f_{occ} = \frac{r_a}{(r_a + r_d)}. \quad (7)$$

The average site occupation at our vacuum level was of 1.63×10^{-1} %, which is to be expected given that, compared to ambient pressure conditions (average site occupation of 100%), the adsorption rate would be magnitudes smaller than the desorption rate.

Table 1: Adsorption and desorption rate and site occupancy at ambient and vacuum pressure.

	Ambient Partial Pressure ($P = 1250$ Pa)	**Vacuum Partial Pressure ($P = 10^{-5}$ Pa)**
r_a [1/s]	4.52×10^5	3.61×10^{-2}
r_d [1/s]	2.21×10^1	
f_{occ} [%]	100	1.63×10^{-1}

From **Table 1**, we expect the membrane to be completely covered by a monolayer of water in ambient conditions, and almost completely uncovered in vacuum. Noteworthy, the desorption

rate r_d is not instantaneous but takes effect in the order of seconds. For a surface that has porosities, we can reasonably expect an even slower desorption rate. This is commonly observed when pumping down vacuum chambers, which may need hours to reach base pressure when they contain porous hydrophilic surfaces (e.g., material evaporators containing evaporated SiN and SiO_2 films).

2.5.2 Adsorbed Mass Induced Drift

The particles of various contaminants present in resonator environment can be adsorbed on the resonator surface, which will directly affect its effective mass. This will ultimately lead to a change of the eigenfrequency (f in Hz), which is linked to the effective mass (m_{eff}) by:

$$f = \sqrt{\frac{k_{eff}}{m_{eff}}}\Big/_{2\pi}, \tag{8}$$

where k_{eff} is the resonator stiffness, in N/m.

In the context of our work, calculations are made to obtain the drift of frequency, in parts per million (ppm), that would be caused by the adsorption/desorption of a monolayer on our silicon nitride resonators. We consider the case of water, as its interaction with SiN is well studied [32], and outgassing of water molecules is known to be a major factor affecting the pressure reached in high vacuum systems.

The equation which allows us to infer the relative frequency change occurring due to the mass variation is the following:

$$\frac{\Delta f}{f} = -\frac{1}{2} \cdot \frac{\Delta m_c}{m_m}, \tag{9}$$

where $\frac{\Delta f}{f}$ is the relative frequency shift, Δm_c is the total mass change due to adsorbates and m_m is the mass of the membrane itself. Finding the mass of the membrane ($m_m = 1.027 \times 10^{-8}$ kg) is

simply done by using its volume (V in m^3) and density (ρ_{SiN} in kg/m^3): $m_m = \rho_{SiN} \cdot V$. Finding the mass of adsorbates requires a few more steps, including estimating the surface density of adsorption sites (S_A in sites/m^2) on SiN. The area of our square membrane (A_m in m^2) is computed with $A_m = L^2$, where L=0.006 m is the length of the side of our square membrane. We also consider the weight of a single contaminant molecule using the equation $m_{cont} = {M_c}/{N_A}$, where M_c is the molar mass of the adsorbate in kg/mol and N_A the Avogadro number in mol^{-1}. For water, $m_{cont} = 2.991 \times 10^{-26}$ kg. Then, the volumetric density of atoms (S_V in atoms /m^3) is computed using the density of SiN and its molar mass (M_{SiN} in kg/mol) as well as the Avogadro number in the equation: $S_V = \frac{\rho_{SiN} \cdot N_A}{M_{SiN}}$. Considering $\rho_{SiN} = 3170$ kg/m^3, we obtain $S_V = 1.361 \times 10^{28}$ sites/m^3. Assuming that each exposed SiN surface molecule is an adsorption site for water, we estimate the surface density of adsorption sites (S_A in sites/m^2) of SiN using:

$$S_A = S_V^{\frac{2}{3}}, \tag{10}$$

which yields $S_A = 5.700 \times 10^{18}$ sites/m^2. The number of contaminant molecules in a monolayer of our device (N_c) is calculated with $N_c = S_A \cdot A_m$. We can finally compute the adsorption mass differential ($\Delta m_c = 6.138 \times 10^{-12}$ kg) using $\Delta m_c = N_c \cdot m_{cont}$.

Using these parameters in equation (9) we have concluded that a frequency drift of about -598 ppm would result from the adsorption of a single complete monolayer of water on both sides of our 90 nm thick membrane the membrane, in ambient conditions. In vacuum, by taking into account the fraction of occupied sites found in 2.5.1 ($f_{occ} = 1.63 \times 10^{-1}$ %), the drift changes to -1 ppm. We therefore expect the frequency of our membrane to increase by 597 ppm during the pump down that brings the membrane from atmosphere to high vacuum.

2.5.3 Surface Tension Induced Drift

The surface tension due to the cohesive hydrogen bonds in an adsorbed water layer [34] on the surface of the membrane can increase the stress of the thin film [35], a parameter on which its resonating frequency depends, as explained in [12] and presented in equation (1). Calculations were carried to estimate the magnitude of the frequency change that would result from the additional surface tension stress.

The equation which allows us to infer the relative frequency change occurring due to the surface tensions stress additive is:

$$\frac{\Delta f}{f} = \frac{1}{2} \cdot \frac{\Delta \sigma_{ST}}{\sigma_{Initial}}, \tag{11}$$

where $\frac{\Delta f}{f}$ is the relative frequency shift, $\Delta \sigma_{ST}$ (in MPa) is the additive surface tensile stress and $\sigma_{Initial}$ (in MPa) is the initial stress of our membranes.

To find the frequency drift caused by this aging source, we follow the same principle as in Stoney's equation [32] in which we use assume that the surface tension of the adsorbates ($\delta s = 73$ mN/m for water) directly adds to the existing stress inside the resonator. In this case, the additive tensile stress of the resonator is given by:

$$\Delta \sigma_{ST} = \frac{\delta s}{t}, \tag{12}$$

where $t = 90$ nm is the membrane thickness. In equation (11) this differential stress is used alongside the initial formation stress of our low stress membranes ($\sigma_{\text{Initial}} = 100$ MPa) to find the frequency drift caused by the water surface tension of a fully covered membrane. The obtained frequency variation, in ppm, for this source is roughly 11,600 ppm for a complete monolayer. Considering our setting with the computed vacuum fractional site occupancy, the drift issued from

this source of stress is evaluated at around 19 ppm. We therefore expect a decrease in frequency of 11,600 ppm when bringing the membrane from atmosphere to high vacuum.

2.5.4 Chemical Stress Induced Drift

The last investigated source of drift issued from contaminant adsorption is the one generated by stresses resulting from chemical reactions between the adsorbates and adsorbent. Similarly to the previously investigated drift source, these chemical reactions will result in an added surface tension on the membrane. There is an extremely large range of potential chemical reactions that could occur, which depend on the composition of the adsorbates and adsorbents as well as the elements present in the environment of the resonator (high vacuum in our case). Fortunately, chemical studies on a silicon nitride cantilever in ambient air have previously been conducted [32] and have revealed possible reactions for this material:

$$\mathrm{SiOH} + \mathrm{H^+} \rightarrow \mathrm{SiOH_2^+},$$
$$\mathrm{SiO^-} + \mathrm{H^+} \rightarrow \mathrm{SiOH},$$
$$\mathrm{SiNH_2} + \mathrm{H^+} \rightarrow \mathrm{SiNH_3^+}.$$

To compute the stress issued from these chemical reactions, Stephan and al opted for a chemical and thermodynamic analysis of surface-free energy to infer a chemically induced surface tension stress [19]. They concluded that, for one monolayer of their SiN cantilever, the site density is of 10^{19} m^{-2}, resulting in a surface tension is of 1.1 N/m. As explained previously and using equation (12) with the values from Stephan's work, it was found that a total chemically induced surface tension stress increment of 12 MPa would occur due to adsorption of water. Using equation (11), this stress increment results in a frequency drift of 175×10^3 ppm for both sides of the membrane.

The fractional site occupancy for our vacuum environment, found in 2.5.1, is factored into the calculated total chemically induced frequency shift to obtain the proper drift for our respective

number of active sites, which is of roughly 285 ppm. Consequently, a decrease of frequency of 175×10^3 ppm is expected when bringing the membrane from atmosphere to high vacuum.

2.5.5 Summary of Surface Drift Effects

The importance of the drift values for the three explored aging sources from sections 2.5.2 to 2.5.4, as well as their magnitude, are in coherence with the results of Stephan's research [32].

Table 2 presents the overall drift expected (-186×10^3 ppm) when bringing our membrane from ambient to vacuum environment. In practice, we expect our measurements to yield smaller drift, since we do not track the frequency during the first few hours of pump down, a critical period where most of the desorption will occur. The measurements begin 24 hours after the vacuum pump down is initiated to allow the vacuum to stabilize at a high value and ensure that the parasitic heat issued from the activation of the piezoelectric actuator reaches steady state, which takes approximately 10 hours (see section 4.1, **Figure 38**).

Table 2: Frequency changes expected from adsorption of water, in ambient and vacuum environment and drift expected when changing form one environment to the other.

Aging Source	**Ambient [ppm]**	**Vacuum [ppm]**	**Drift from Ambient to Vacuum [ppm]**
Mass Change	-598	-1	597
Water Tensile Stress	11,600	19	$-11{,}600$
Chemical Tensile Stress	175×10^3	285	-175×10^3
Total Aging of Above Sources	186×10^3	303	-186×10^3

2.5.6 Mounting Aging

Mounting methods employed to fix the resonators can have an impact on the aging of the device as well as its *Q-factor*, as explained in section 2.2.2. Designing mounts requires delicate engineering and consideration of many factors such as the environment of operation of the device, the type of material to which it will be bonded, physical dimension restrictions, ease of maintenance as well as interference with instruments used.

The materials used to create the mounts will suffer from a lot of the same aging sources as the resonator itself. They will deteriorate through time and will outgas their own contaminants, which will be fed into the environment of operation. More importantly, as the mount ages, the external stress that it exerts onto the resonator itself will be changing which will directly impact its frequency [21]. Mount materials will usually be subjected to very long-term changes we consider as creep. As a result, we suspect that the aging effect it will have on the resonator will take form of a slow and low amplitude parasitic drift which is hard to detect and suppress entirely. Thus, to ensure the measures exclude potential drift from the mount's aging, it is preferable to use low-aging materials when designing it.

Additionally, some mounting mechanisms can amplify the drift effects caused by thermal expansion by restricting the expansion of the device. We previously conducted theoretical work, found in appendix D, on the stress variation caused by the difference of thermal expansion between the substrate of our device as well as its membrane. An athermalization concept was suggested to suppress frequency drift by bonding the substrate to a plate that would cancel the effect of substrate thermal expansions. While this methodology was not implemented in practice, our expertise was used to design the current mount (3.1.3.1) and prevent it from restricting thermal expansion of the substrate, which would have worsened aging induced by the mount.

3. Experimental Setup

In this work, an experimental setup is used to measure the long-term frequency drift of one of our SiN membranes. The simplified system is represented in **Figure 4** below. An optical system is used alongside a lock-in amplifier to read the frequency of the vibrating membrane. Additionally, the temperature of the thermistor placed inside the vacuum cell is obtained with a multimeter through the measurement of its resistance. The data acquired by the lock-in amplifier and the multimeter is streamed directly to a computer where they will be saved as text files with a Python script. More scripts will then be used to interpret and manipulate the data, ultimately allowing us to extract the results.

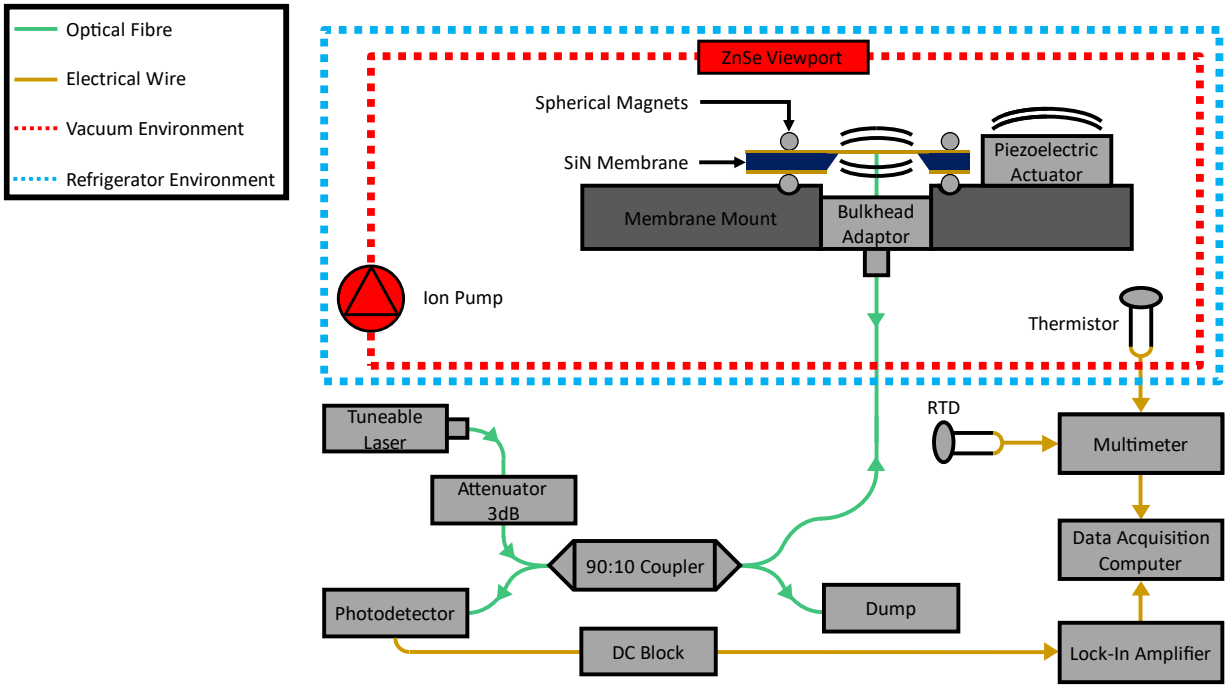


Figure 4: Simplified experimental system schematics.

3.1 Membrane

The studied membranes, seen in **Figure 5** below, are fabricated in our laboratories, using purchased silicon wafers with either low-stress or high-stress silicon nitride coated by low pressure chemical vapour deposition (LPCVD). The silicon frames, which have thicknesses of 525 microns, serve as supports for our square membranes with widths of 6 mm and thicknesses of 90 nm. The difference between the high-stress (stochiometric Si_3N_4) and the low-stress (silicon rich Si_3N_X where x<4) SiN is that the latter, has a smaller ratio of nitrogen to silicon than the former which is at the stochiometric ratio. As a result, the internal stress of the membrane will be of 100 MPa and 1000 MPa respectively for the low and high-stress SiN. This work uses a low-stress SiN membrane, which is more robust but less sensitive than its high-stress counterpart.

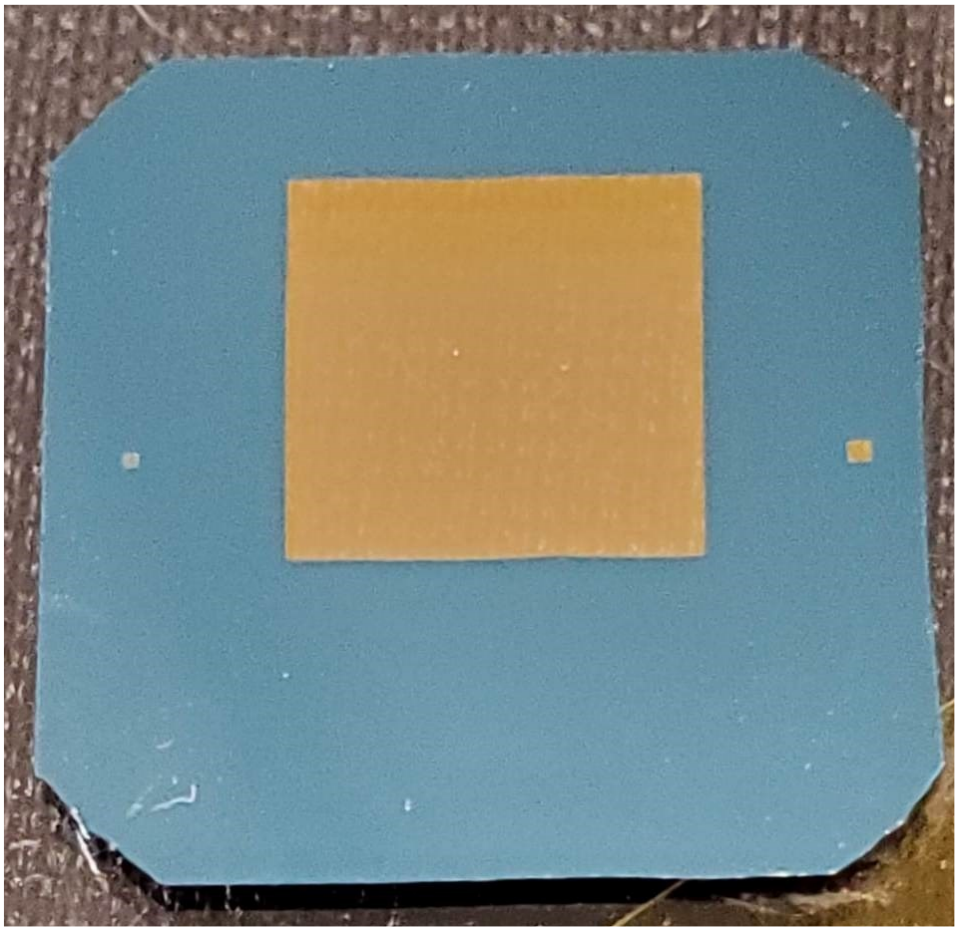


Figure 5: Plain 6 × 6 mm low-stress SiN membrane used in this work.

3.1.1 Fabrication Process Overview

The fabrication process of our membranes, which has been extensively documented in other work from our lab [36], takes place at the University of Ottawa facilities. The steps mainly consist of:

1) Spin coating photoresist on a double-polished 525 μm silicon wafer coated with SiN,
2) Aligning a photomask and exposing the nitride on one side of the coated wafer, through photolithography,
3) Etching the pattern on the exposed SiN with reactive ion etching,
4) Using hot KOH to etch the silicon substrate on the previously created pattern, which will structurally release the SiN layer and create the desired membranes.

The previously designed photomask [36] did not focus on the production of the membrane sizes used in this work, thus had to be extensively reworked. *Tanner L-Edit IC* [37] was used to create many different iterations of a new photomask, until a satisfactory pattern was achieved. The main modification over the previous design is that the production of 12×12 mm membranes was discontinued, thus allowing this photomask design to focus on the production of clustered 1×1 mm, 1.5×1.5 mm, 3×3 mm, and 6×6 mm membranes. The density of the produced membranes would also be increased by filling in some previously unused space with modified compact versions of the 1.5×1.5 and 3×3 mm membranes.

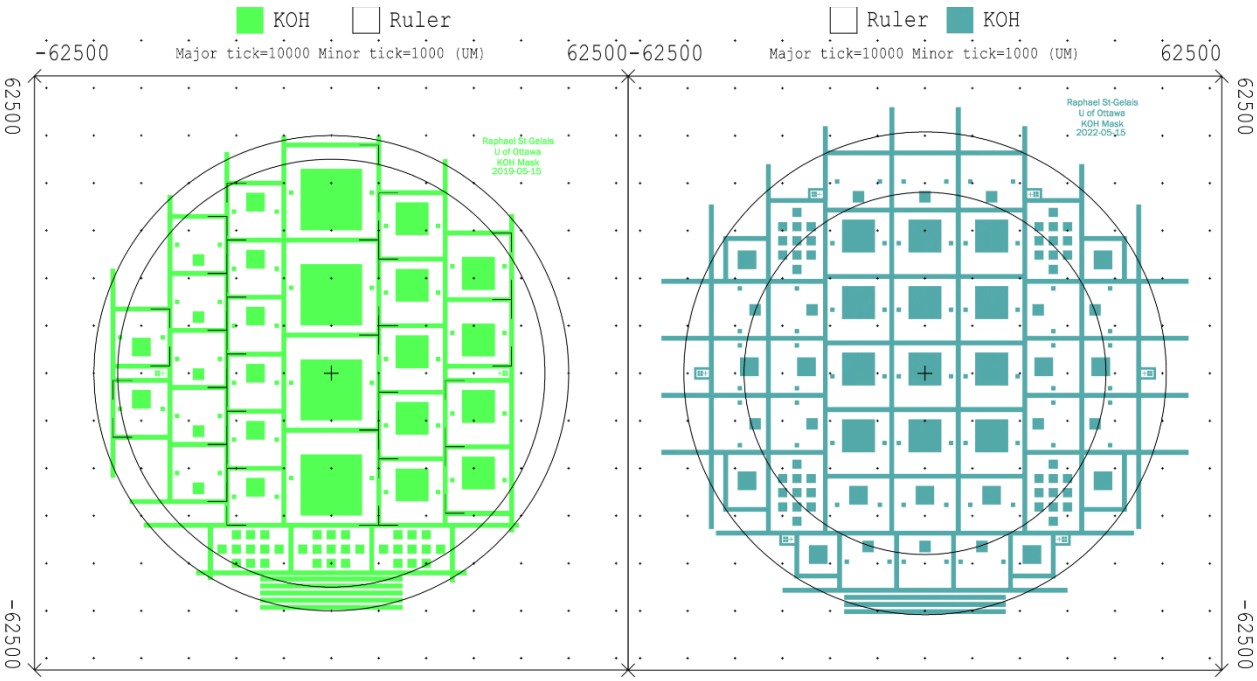


Figure 6: Old (on the left) and new (on the right) photomask designs.

3.1.2 Adhesive Membrane Mounting (Superseded)

As we explained in sections 2.5.6, the mounting mechanism was carefully designed to limit aging, improve the *Q-factor* of our device as well as enhance the assembly process to make it easier and faster. The previous mounting methods used in our lab relied on the use of adhesives such as *Crystalbond*^{TM}, nickel paste or double-sided tape. When developing the new magnetic mounting method, the flaws of these old mounting techniques were addressed while also introducing other general improvements. The principal flaws shared by these adhesive based mounting methods are:

- The adhesives consist of polymers and other agents that deteriorate and age rapidly, which falsifies membrane aging measurements,
- The adhesives restrict the thermal expansion of the membrane, which affects its stress,
- The adhesives have a very irregular and large bonding area which greatly increases the damping of the membranes and impedes their *Q-factor*.

3.1.2.1 *Crystalbond*^{TM} Mounting

The *Crystalbond*^{TM} used in our lab is a temporary adhesive that bonds to different materials such as glass, metals, and ceramics. This glue is solid at room temperature, but it can easily and quickly be melted at around 120 °C [38] to reach its flow point and quickly cures when left to cool. The bond is said to be temporary since it can easily be dissolved with acetone.

The mounting of our device and its disassembly is briefly explained below. Solid fragments of the adhesive are applied on the mounting plate we wish to bond our membrane on. The mounting plate is placed on a hot plate that is heated above the melting point of the adhesive. Our membrane is then placed and easily positioned in the puddle of melted adhesive using tweezers. The membrane is held in place while the *Crystalbond*^{TM} cures after we remove the mounting plate from the heat source. When cured, the bond is extremely solid, but it can quickly be undone. The preferred disassembly method used in our lab is to put the mounting plate, with the membrane still fixed on it, back onto the hot plate. The membrane is then held with a tweezer while the adhesive liquifies. Once the adhesive is melted, the membrane is removed and placed in a receptacle to be cleaned. The hot plated is turned off and the *Crystalbond*^{TM} residues on the mounting plate and the membrane are cleaned using acetone.

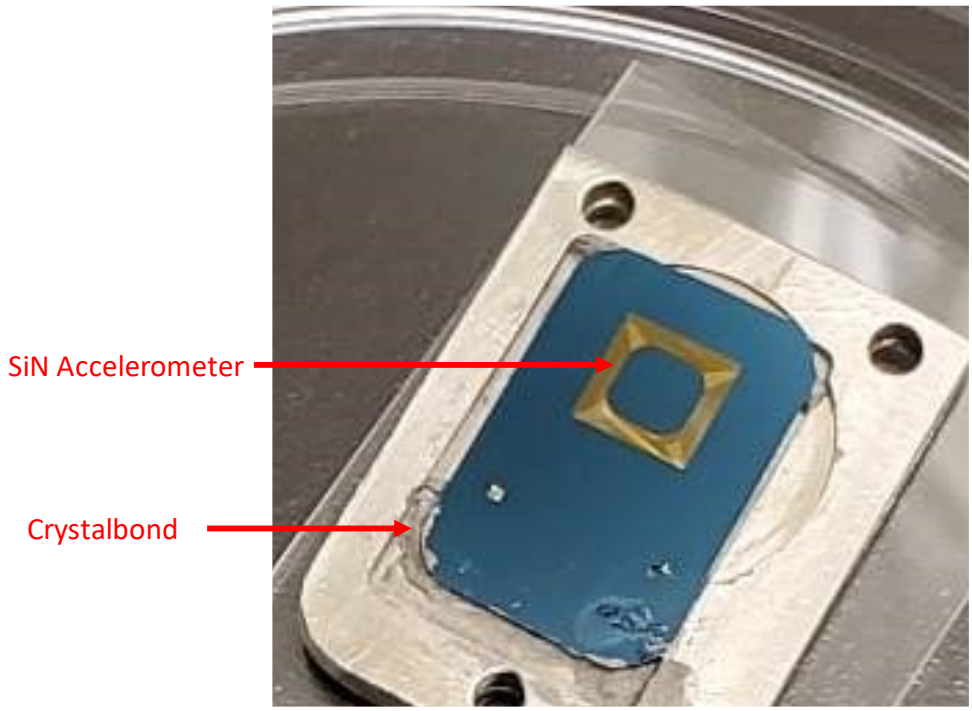


Figure 7: *Crystalbond*^{TM} mounted SiN accelerometer.

While *Crystalbond*^{TM} offers a great ease of installation, it suffers from significant problems. The low melting point of this adhesive causes issues when used in situations requiring high temperatures. An example is the baking of the vacuum system to improve the pump down speed and remove bulk contaminants [39]. If baking is conducted at temperatures higher than the melting point of *Crystalbond*^{TM}, the bonds will be undone and adhesive vapours will be introduced in the cell. Another example is our heat transfer experiments, where heat was found to soften the adhesive and make the membrane fixation unstable [40].

3.1.2.2 Nickel Paste Mounting

This adhesive is a silica paste containing nickel [41], which allows components to be bonded together while also being thermally and electrically conductive. The main advantage it provides, compared to the other adhesive bonding methods, is that it has a very high operating temperature that goes up to 538 °C [41], while also allowing electrical connections to be made

between components. Its principal drawback comes from the fact that it has a very long curing time ranging between 4 to 6 hours, thus greatly reducing the mounting efficiency. While it is soluble in water, it is much messier and harder to clean than *Crystalbond*^{TM}.

To mount membranes with this agent, it must be mixed extensively. A thin coat of the paste is then applied to the mounting surface. The membrane is placed and easily positioned in the deposited paste. It is then left to cure to ambient air from 2 to 4 hours followed with an additional 2 hours of curing on a hotplate at 93 °C [41]. After this extensive period, the nickel paste has fully solidified and formed its bond. Disassembling the mounted membranes involves higher breakage risks than when using *Crystalbond*^{TM} , since dissolving the fully cured nickel paste with water is harder and requires more aggressive manoeuvres. This results in a much dirtier surface after dissassembly.

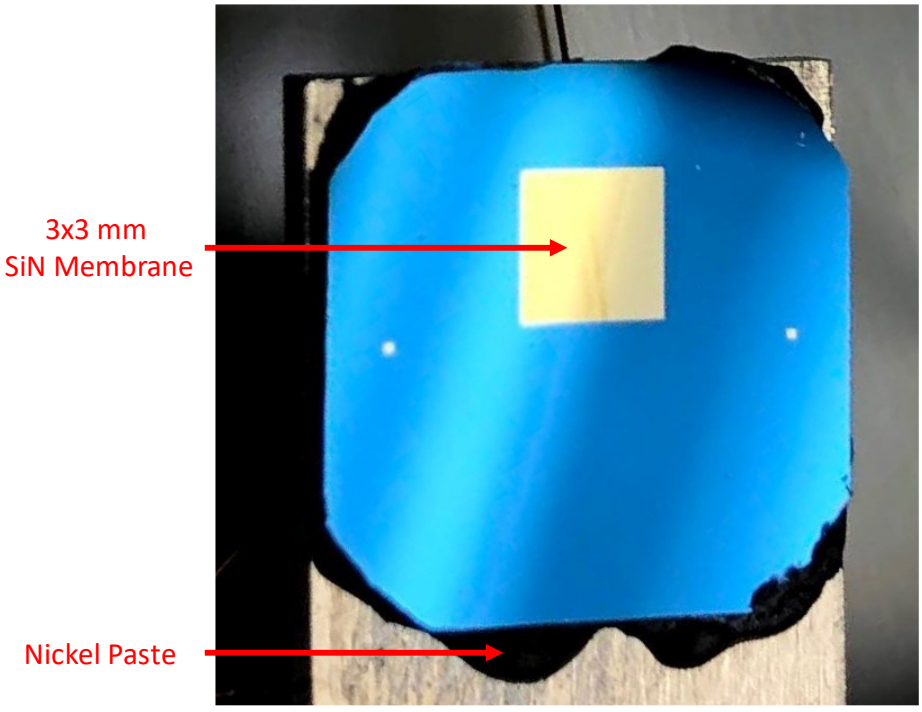


Figure 8: Nickel paste mounted 3 × 3 mm SiN membrane.

3.1.2.3 Double-sided Tape Mounting

The last adhesive mounting method used in our lab was double-sided tape. This method was by far the quickest and easiest among those used. A strip of tape was simply cut to the desired size and placed on the mounting surface. The membrane was then carefully positioned on the tape using tweezers and pressure was applied on the bonding region to ensure it adhered well. To remove the membrane, a tweezer was used to tightly pinch its frame and repeatedly lightly rock the membrane from side to side in a shear motion to detach it as softly as possible.

While this method was the simplest by far, it was also the least used because of its disadvantages. Tape was never used in situations where temperature would rise to high values due to its nature. While the position in which the membrane sits could easily be modified before curing with the *Crystalbond*^{TM} or nickel paste, it wasn't the case with the tape which would instantly stick to the membrane and render any minimal repositioning extremely hard. Additionally, dismounting the membrane from the tape could result in an abrupt release, leading to the breakage of the membrane. The presence of tape also strongly damped the mechanical resonance, which greatly reduced the performance of the device.

Figure 9: Double-sided tape mounted *Norcada* SiN membrane.

3.1.3 Magnetic Mounting

The new developed method of mounting relies on pinching the membrane between two layers of magnets. This mounting technique addresses all the disadvantages noted from the previous adhesive methods while also greatly improving the *Q-factor* of our device since all parts involved are rigid, which provides low damping. To install a membrane with this method, a layer of magnets is placed on a magnetic mounting plate. Using a tweezer, the membrane is carefully placed on these magnets, which elevates it from the surface of the mounting plate. Based on our various tests, the best magnet disposition is a triangular configuration of 3 small magnets. It provides the best balance between damping minimization and membrane installation stability. Finally, to secure the membrane in place, the next layer of magnets is carefully superimposed onto the membrane, thus pinching it by means of the magnets' attraction forces.

This mounting method greatly improves the previous flaws. The use of metals in the mounting assembly guarantees low deterioration and aging of the mount in comparison to the superseded adhesives. This ensures aging measurements of the membrane with minimal impact from the clamping system. When it comes to ease of installation, this method is by far the most efficient. It takes a few seconds to lay the first layer of magnets with a tweezer and successfully sandwich the membrane. Undoing the mounting is as simple and quick. Simply use tweezers to remove the upper layer of magnets and then the membrane. This allows for instant installation of membranes with very low probability of destruction induced by the mounting. Furthermore, there are no residual bonding agents or mess which creates a clean mount assembly where contaminants are minimized. Although the membrane is well pinched, lateral motion is still possible. This ensures that the membrane can be easily repositioned in the desired location by simply pushing on the appropriate side with a tweezer. Additionally, the lateral thermal expansion of the membrane

is not restricted. The magnets used are nickel coated with a neodymium core which allows them to be used in a very high temperature setting. The chosen rare earth neodymium magnets are of N52 grade and have an individual approximate pull force of 0.5 lb. The preferred magnet size ranges between 0.5 and 1 mm and the spherical or disc shape is optimal. The magnets can be precisely located on any desired part of the magnetic mounting plate meaning that the area where they apply their pinching force on the device we wish to retain is known and controlled.

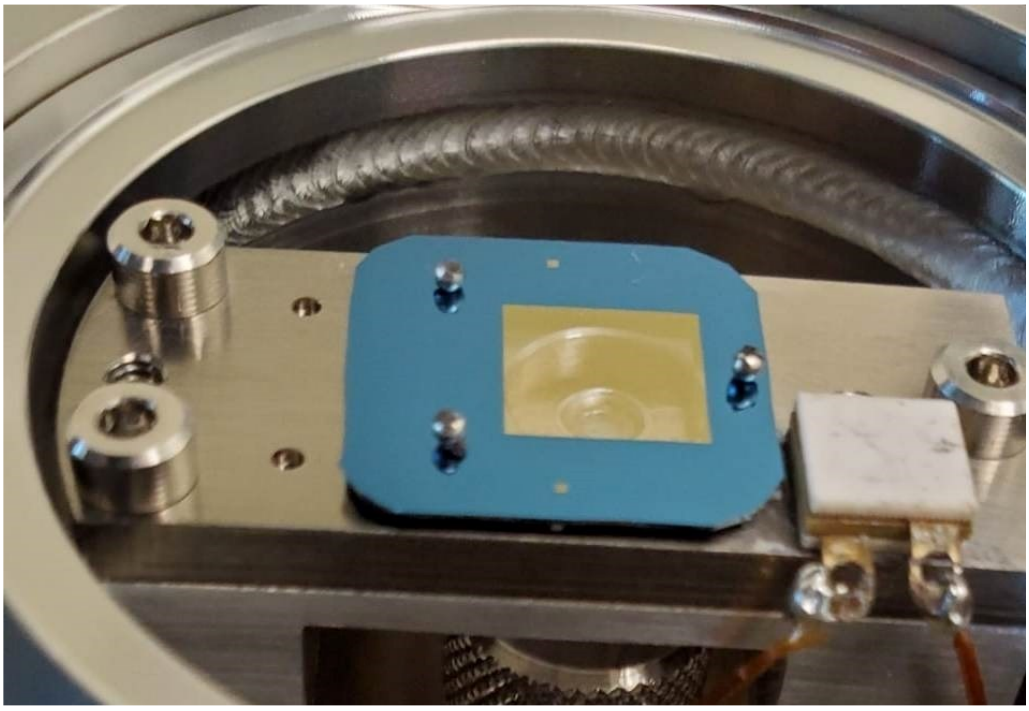


Figure 10: Magnetic membrane mount assembly with accessories in vacuum cell.

3.1.3.1 Designed Magnetic Mount

The mounting method used throughout the aging measurements was designed with many features. The first part in the mount mechanism is the bottom plate pictured in **Figure 11**, which is made from 6061 aluminum. There are two versions of the bottom plate: one features threads for a *ThorLabs HAFC bulkhead adapter*, the fibre interferometer used in the aging measurement, while the other one is threaded to accommodate a *ThorLabs F230APC-1550* collimator. Both

versions feature five threaded M2 holes where three are used to put retaining screws that will secure the top plate and bottom plate together. The two other holes are used for the installation of guiding rods. The rods, illustrated in **Figure 13**, are used to easily guide the mount down into the narrow and cylindrical vacuum cell, where there is insufficient clearance for our hands. This greatly facilitates the installation of the mount inside the cell and tremendously reduces the risks of pushing too hard and breaking or shifting the fragile components. Additionally, the plates feature a rounded edge with a large radius to ensure the mount rides on the inner walls of the cell efficiently. On its other end, the bottom plate is threaded with two 6-32 holes intended to accommodate spring plungers, as seen in **Figure 12**, to ensure that the mount applies pressure on the inner walls of the cell and essentially wedges itself into position inside the cell.

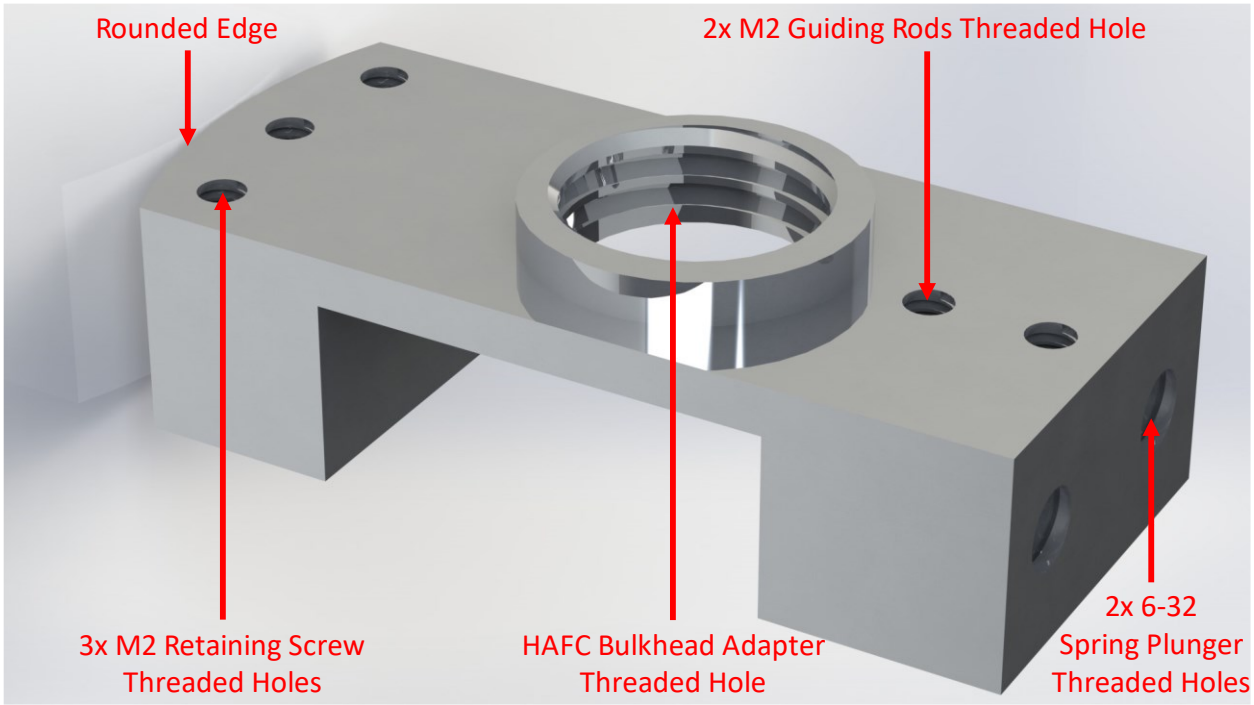


Figure 11: HAFC bottom plate CAD rendering with captioned features.

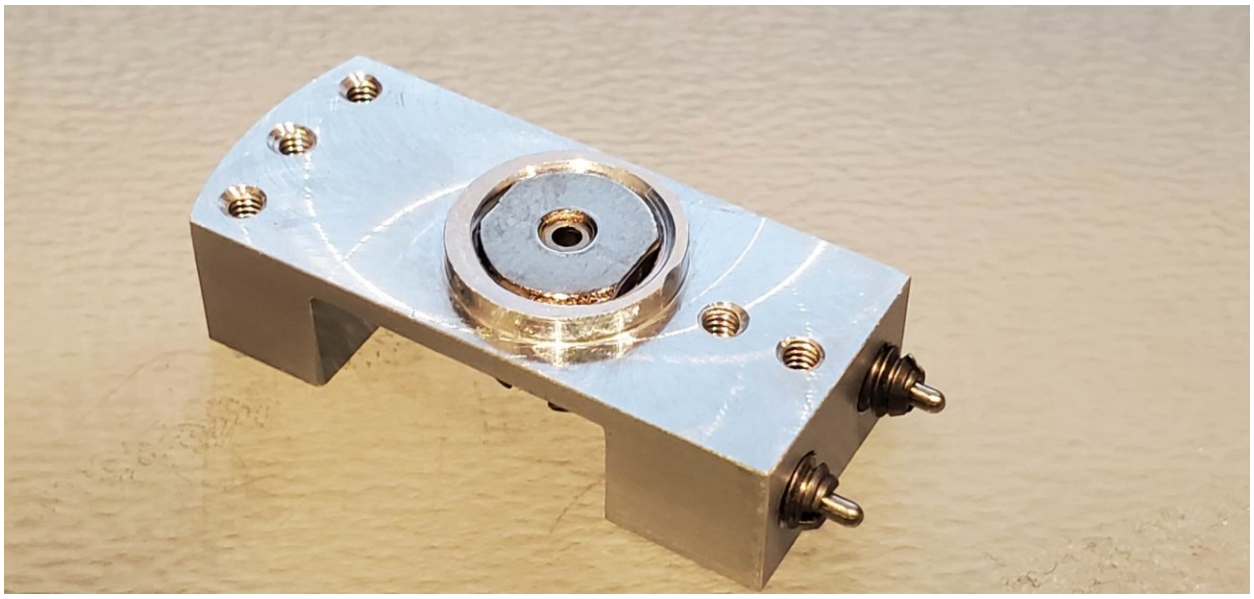


Figure 12: HAFC bottom plate machined part with spring plungers and HAFC bulkhead adapter.

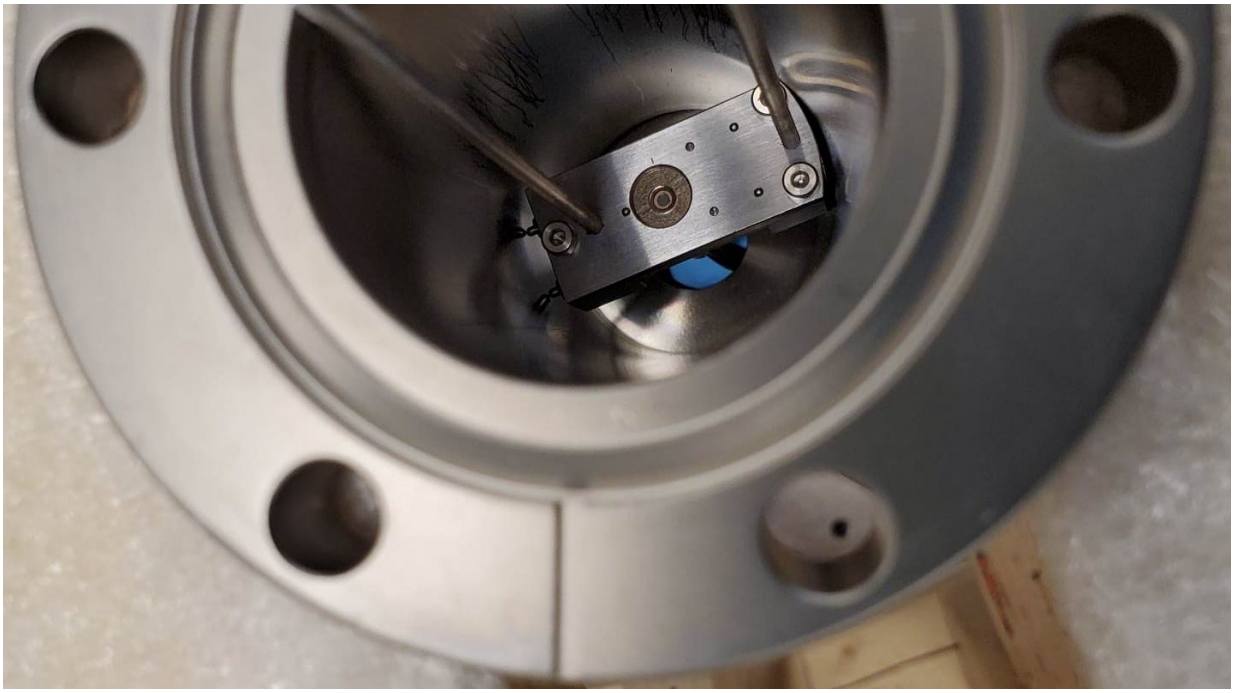


Figure 13: Magnetic mount with guiding rods inside vacuum cell mounting nipple.

The next core part of this mount design is the top plate. The material for this plate needed to have high magnetism to ensure that magnets would easily adhere to its surface, hence why low carbon steel was selected. The magnetic top plate, depicted in the **Figure 14** features precisely

positioned spherical divots with a depth of 0.5 mm and a diameter of 1 mm. These small holes act as reception sites for the lower layer of 1 mm of diameter spherical N52 neodymium magnets, as depicted by **Figure 15**, to assure that they are well seated and restrict their unwanted movement. Instead of having only three divots, an additional row was added. These lower holes allowed us to mount our mass loaded SiN accelerometer, which has a substantially bigger height than our membranes. The top plate, like its lower counter part, has two threaded M2 holes at the front and back to screw in long guiding rods, while also featuring a large radius on one of its edges to ensure it slides smoothly inside the cell. Additionally, an optical opening is included to eliminate potential blockage of the optical path for the HAFC bulkhead adapter and the collimator. Finally, the magnetic plate features three clearance holes allowing M2 retaining bolts to screw into the bottom plate and secure the two parts together.

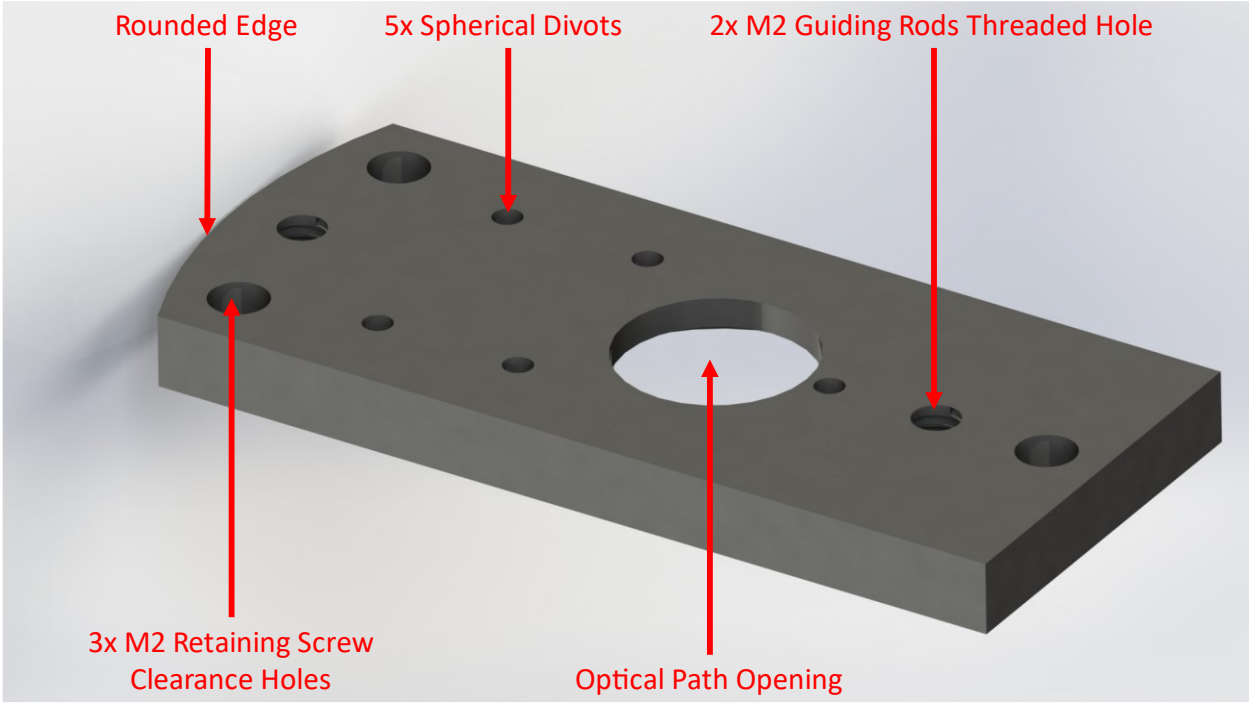


Figure 14: Magnetic top plate CAD rendering with captioned features.

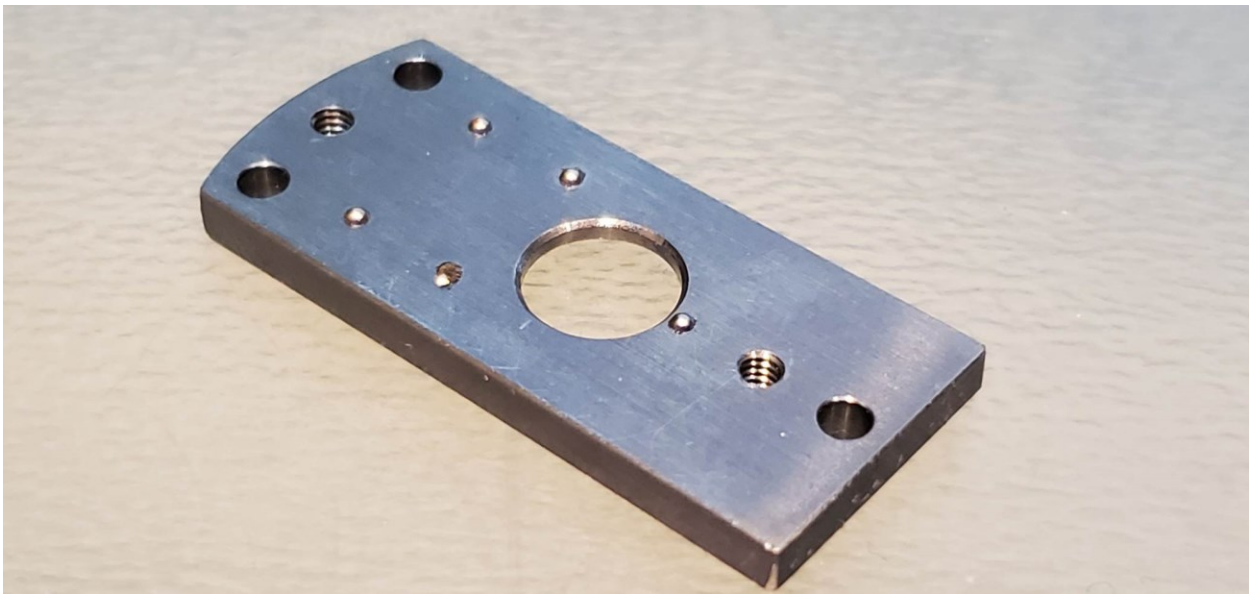


Figure 15: Magnetic top plate machined part with lower layer of spherical magnets.

With the design complete, the parts are machined at the mechanical engineering department machine shop, and the mount can finally be assembled. The magnetic top plate is screwed on the bottom plate with the retaining screws, while the spring plungers are threaded in their respective position. The plungers can be fine-tuned by slightly screwing them outwards or inwards until the achieved tightness in the vacuum cell's opening is achieved. The next step consists of installing the piezoelectric actuator on the top plate. Due to the baking procedure the vacuum cell undergoes, the piezoelectric actuator is bonded to the top plate with nickel paste. The next step is to place magnets of the lower layer into the desired divots of the magnetic plate. The membrane can then be softly deposited onto the first layer of magnets using a tweezer. Finally, the upper magnets are gently added one by one on top of the membrane, while trying to approximately superimpose them to their matching lower counterpart. If done correctly, the magnets will slightly shift and align themselves right above their lower counterpart. It is important to gently deposit the top layer of magnets since the magnetic pull force applied by the small spheres is quite large. If not calmly controlled, the loose upper magnet can snap onto the location of its lower counter part and collide

with the membrane with such force that it may displace its frame. Thus, one must ensure not to recklessly or loosely handle these magnets. Once the membrane is fully pinched between the two layers of magnets, it can be slightly realigned or repositioned by pushing on the side of its frame using a tweezer. Finally, the threaded guiding rods are inserted into their holes and the whole mount assembly can be pushed down to the desired location within the vacuum cell.

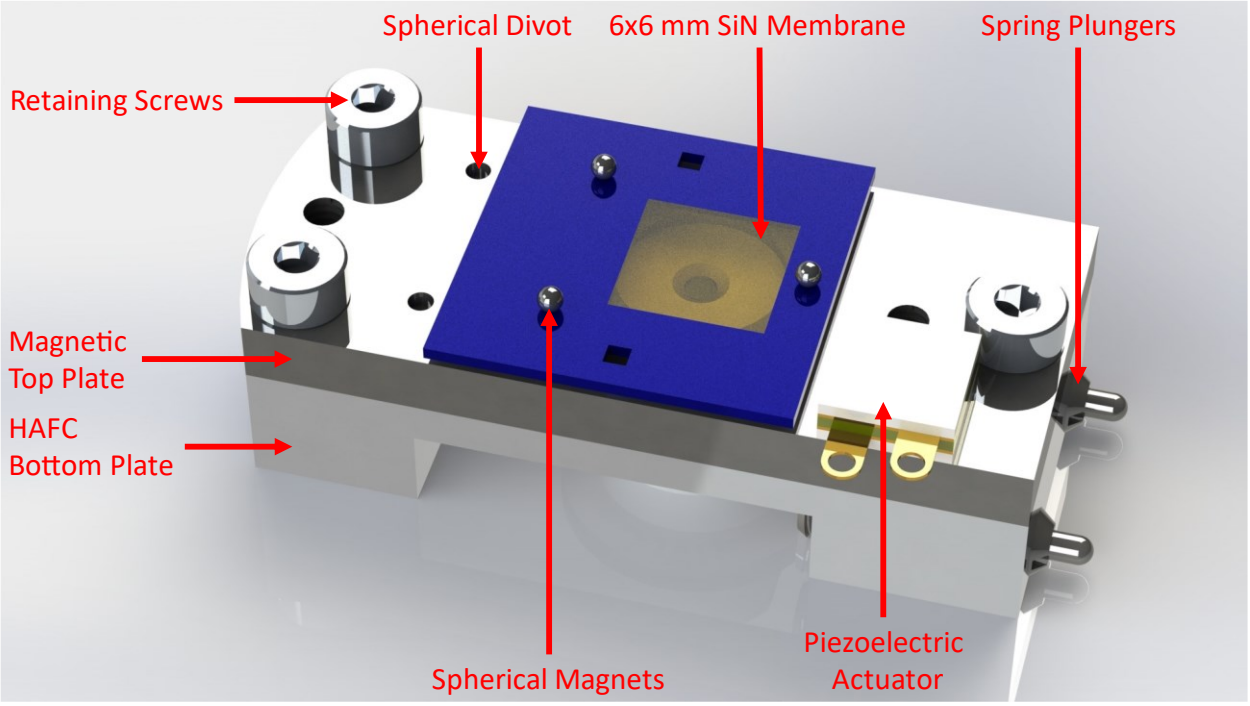


Figure 16: Magnetic membrane mounting design CAD rendering with captioned features.

This mounting design proved its efficiency, throughout the long period of measurements as well as the multiple times the vacuum cell was serviced. The design allowed us to reassemble and dismount our device with such ease that the number of membranes wrongly manipulated or broken during these steps were drastically lower than any of the adhesive based mounting methods. Additionally, the simplicity of this method saved us a lot of assembly time allowing us to focus on various other tasks. Since this design has proven to be greatly successful, a kinematic variation of the top plate has been designed to add flexibility when it comes to the placement of the lower layer

of magnets. This new magnetic top plate retains the same features of the previous version but replaces the fixed spherical divots with a large recessed magnet installation surface, as seen in **Figure 17**. With this feature, disc magnets can be positioned with much more flexibility, which will allow us to use membranes of different shapes and frame sizes. The magnets could also be positioned in a manner where the MEMS can be installed with lateral offsets so that the laser points at non-central parts of the SiN membrane such as its corner. The use of disc magnets also eliminates potential slipping and movement that the spherical magnets could bring.

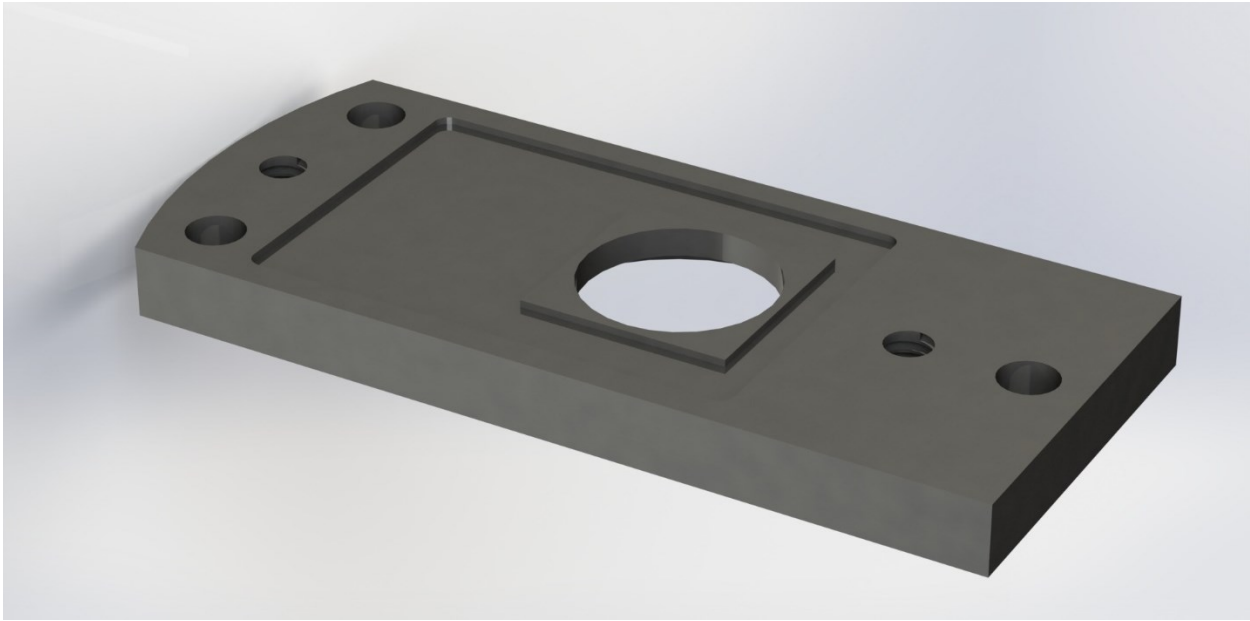


Figure 17: Kinematic magnetic top plate CAD rendering.

Systematic testing wasn't conducted to establish the exact *Q-factor* enhancement induced by the magnetic mounting method. It would be insightful to test a singular membrane under all the different mounting methods to establish a direct comparison of the obtainable Q. Although this data is not yet available, we have recorded *Q-factor* measurements on different membranes with a few methods. Overall, the magnetic mounting seems to greatly enhance the achievable *Q-factors*, especially at low resonance eigenfrequencies where membrane modes are more strongly coupled to the silicon frame [12]. The highest achieved Q on mode 1-1 for our 6×6 mm low-stress

membranes with adhesive and magnetic mounting was respectively of 150,698 and 2,089,066. Similar data from other work in our lab revealed that magnetic mounting enhanced the *Q-factors* for the smaller membranes as well as our accelerometer which was able to achieve a value 20 times larger [14], [42], [11].

Quality-factors (Q) are calculated by using the following equation:

$$Q = \pi \cdot f \cdot \tau, \tag{13}$$

where τ, in s, is the vibration decay rate obtained from an exponential curve-fit of the ring-down of our resonator and f is its resonance frequency in Hz. Ring-downs are taken by using our *Zurich MFLI* lock-in amplifier to excite our membrane and continuously record the voltage amplitude of its vibrations from the moment it stops being driven by the piezoelectric actuator to the moment it stops vibrating on its own. Essentially, the longer it takes for the resonator's vibrations to die out, the larger the decay rate will be, thus providing a higher *Q-factor*.

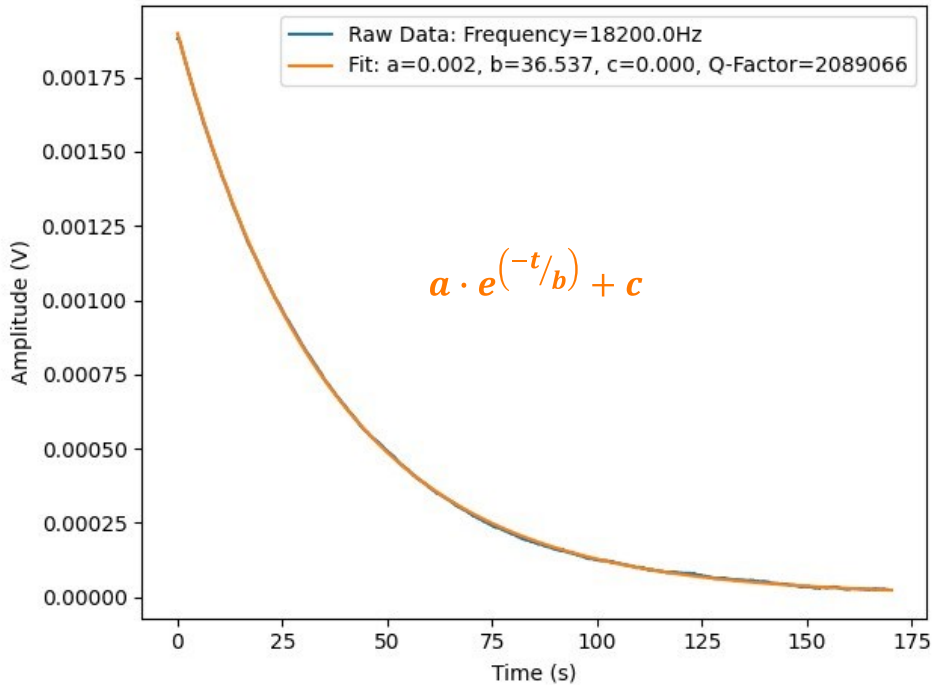


Figure 18: Low-stress 6 × 6 mm SiN membrane mode 1-1 ring-down curve fit.

3.2 Environment of Operation

The environment in which the membrane resonates, displayed in the **Figure 4**, will have an impact over its *Q-factor* (explained in section 2.2.1), as well as its frequency thermally induced drift. Thus, our MEMS device is installed inside a cell under high-vacuum that is placed in a temperature-controlled air bath to maximize the Q and greatly minimize the thermally induced frequency drifts.

3.2.1 Vacuum Environment

The vacuum cell, shown in **Figure 19**, was designed to be a portable chamber that can hold high vacuum levels of 10^{-8} Torr, ensuring a high *Q-factor* can be reached while eliminating heat transfer by convection which adds temperature stability. The main limitation to rendering it portable was the restriction imposed by powering and transporting such a large vacuum pump. This is why a small form-factor ion pump was chosen to sustain the vacuum in this cell. A turbomolecular pump, *Edwards T-Station 85* in our case, must be used to initiate the pump-down of the cell. Once a vacuum level around 10^{-5} Torr is reached, the ion pump valve is opened and the turbomolecular pump valve is closed to let the *Gamma Vacuum 3S Titan DI* ion pump take over. The mounting nipple is a long cylinder containing the components of our setup such as the magnetic mount assembly presented in section 3.1.3.1 and a *Measurement Specialties 55036 Glass NTC* calibrated thermistor. An optical feedthrough at the bottom of the cell allows the external optical fibre to be connected to a vacuum compatible fibre within the chamber. This vacuum compatible fibre, with a polished connector (PC) tip to angle polished connector (APC) tip, is then connected to the HAFC bulkhead adapter allowing the laser signal to be brought to the membrane. The cell is also designed with a zinc selenide (ZnSe) viewport above the mounting nipple, to conduct radiative cooling experiments. An electrical feedthrough was also fitted to the cell to allow

external wires to be connected to the piezoelectric actuator and thermistor present inside the mounting nipple.

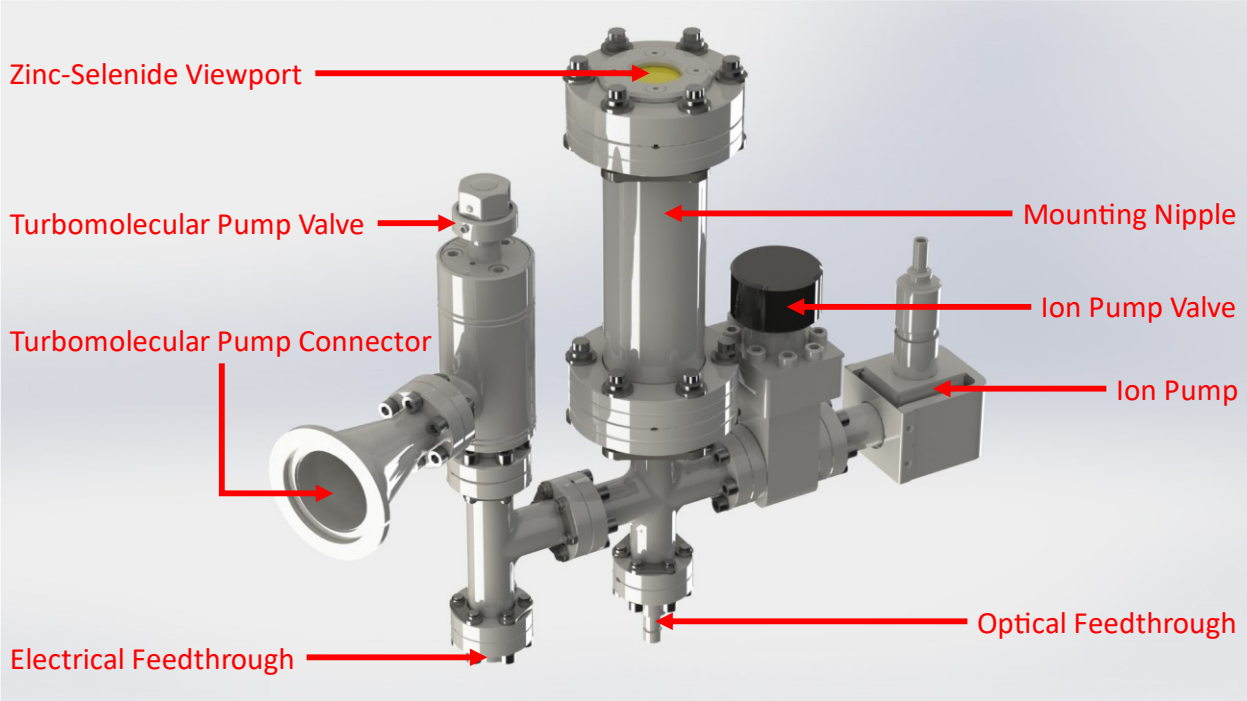


Figure 19: Vacuum cell CAD rendering with captioned features.

3.2.2 Air Bath Environment

While placing the membrane under vacuum already greatly reduces thermal fluctuations by suppressing convective heat transfer, it is not enough to ensure a stable environment, hence why an air bath is required. This device, presented in **Figure 20**, provides a small, enclosed environment in which air is maintained at a very stable temperature. Our experiment uses a *Measurement International 9300* temperature-controlled air bath, with a temperature stability of ± 50 mK for a respective change of ± 1 K in ambient temperature of the lab. Cooling and heating in this unit is achieved using a combination of two Peltier devices controlled by a PID to reach the desired temperature set by a potentiometer. To ensure that the unit is not constantly alternating between a heating and cooling regime, the desired temperature in the air bath (around 23.9°C) was

set to a value that is far from the achievable lab temperature (around 21°C) while considering the HVAC (heating, ventilation, and air conditioning) system changes in winter and summer. Additionally, this air bath features a digital display of its internal controlled temperature as well as a wire feedthrough side port with a removable insulating foam seal.

Figure 20: Air bath containing vacuum cell.

3.3 Frequency Measurement System

Measuring the frequency of the actuated fundamental mode of our membrane and analyzing its drift over a long period of time allows us to quantify the aging of our device. We must also measure the temperature inside the vacuum cell to artificially compensate and remove unwanted thermally induced drift, to which our membrane is very sensitive. The instrumentation

used to measure frequency of our membrane and the cell temperature is explained in the following sub-sections.

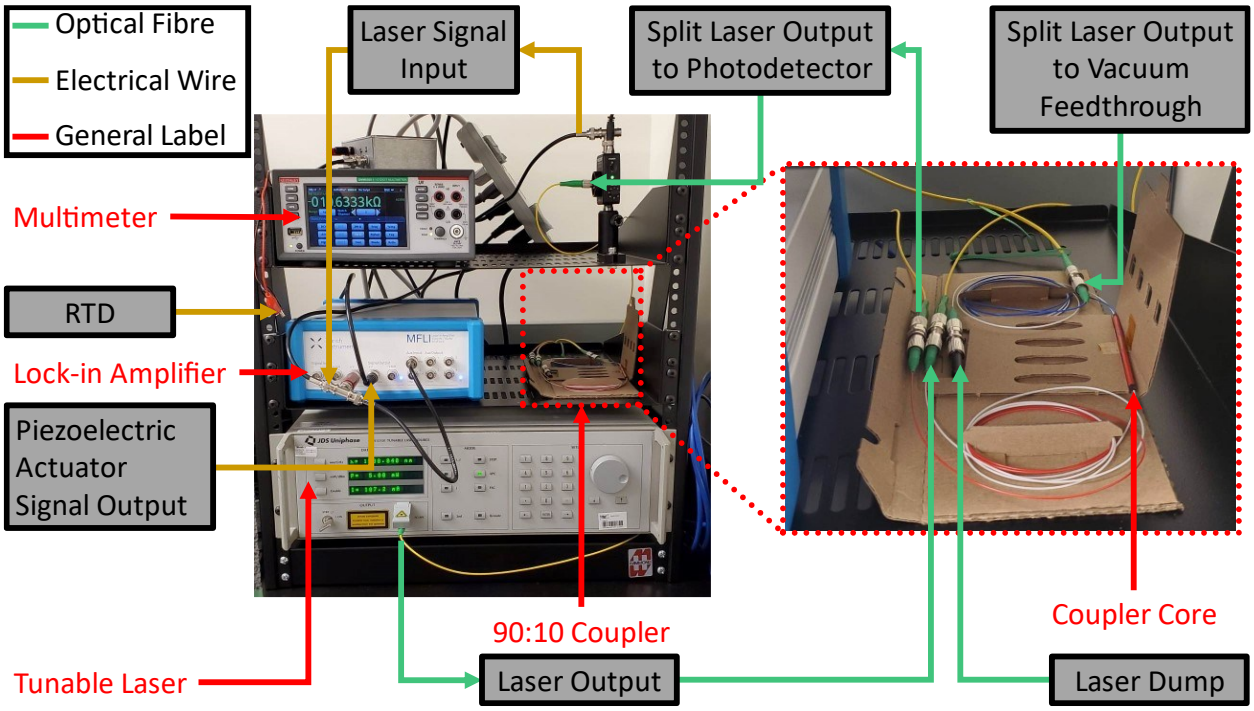


Figure 21: Instrument system setup.

3.3.1 Optical System Basics

Obtaining the frequency of our resonator is ultimately achieved by measuring the optical signal's voltage converted from light interference occurring in the optical system, a concept largely established and explained in Rugar's work [43]. The optical system is comprised of a tunable laser (*JDS Uniphase SWS15101*), a 90:10 fibre coupler (*Thorlabs*), a switchable gain photodetector (*Thorlabs PDA20CS2*) and a lock-in amplifier (*Zurich MFLI*). The tunable laser is used to output single-mode laser light with a precisely controlled wavelength, current and power. The attenuator's goal is to cut down the laser's power to ensure that it does not heat our membrane and falsify our results. However, an additional study we conducted proved that it was not required. The objective of the coupler is to redistribute the laser light to the appropriate parts of our optical system. The

role of the photodetector is to convert the received laser light intensity into an oscillating voltage signal that is sent to the lock-in amplifier. The latter is used to extract the frequency from the voltage input signal, supply an output voltage for our piezoelectric actuator (to excite our membrane) and then continuously track our resonance frequency.

The effect of the laser's power on the frequency of the membrane was tested by installing a *Thorlabs V1550A* variable attenuator. The voltage of this device was controlled by our lock-in amplifier and was swept bidirectionally from 0 to 5 V, which would respectively attenuate the laser power by 1.5 to over 25 dB. The recorded membrane frequency variation was around 4 Hz (200 ppm), as seen in **Figure 22**. Since the tunable laser has an hourly peak-to-peak laser power fluctuation of around 0.01 dB, we thus expect an hourly frequency shift of around $\pm$0.085 ppm during our total period of measurement. Another test we conducted studied the frequency variation as a function of the fringe's signal. To vary the fringe's signal, the tunable laser's sweep function was used to change the wavelength with steps of 0.015 nm from the minimum to the maximum of the fringe's voltage. As we can see in **Figure 23**, the frequency only fluctuates by about 0.5 Hz (25 ppm) during this study. With these tests, we were able to understand the effects of laser power and fringe drift on the frequency of our device. Thus, to ensure that we minimized their impact we implemented a fixed 3 dB attenuator between the tunable laser and the coupler.

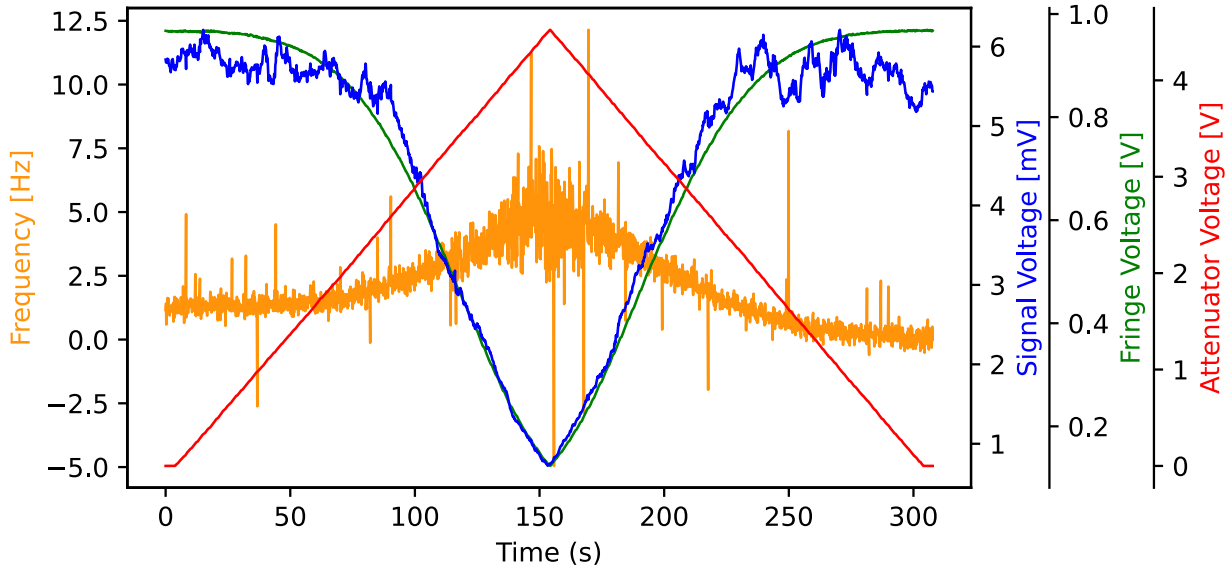


Figure 22: Frequency variation of membrane as a function of laser power.

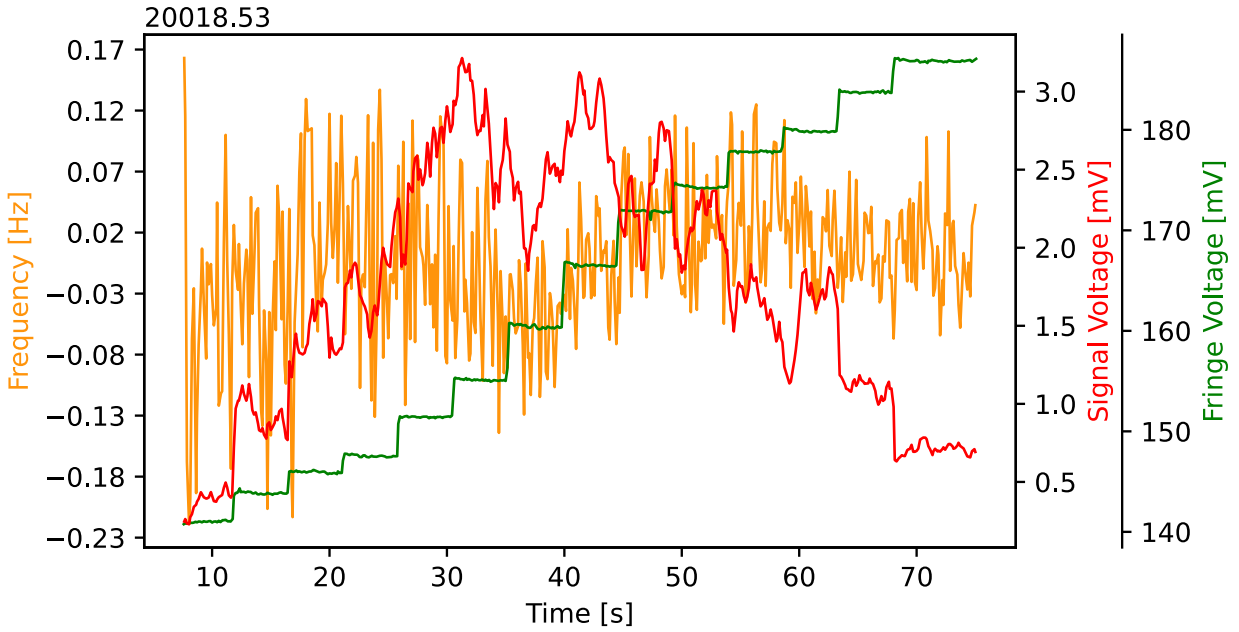


Figure 23: Frequency variation of membrane as a function of fringe voltage.

The optical path followed by the laser is explained and broken down in **Figure 24**. The laser signal comes out of our tunable laser at a power of 5 mW and continues to pass through a 3 dB attenuator. The laser then enters a 90:10 coupler where the signal is split; 90% of the attenuated laser power is dumped while the remaining 10% continues towards the membrane. As the incident laser light reaches the end of the fibre in the bulkhead adapter, around 4% of the light will be

internally reflected by the tip. The remaining light will shine on the oscillating membrane whose distance to the fibre tip (*d* as identified in **Figure 24**) is constantly varying. The light reflected from the membrane is redirected back into the fibre. The reflected light sources, as observed in **Figure 24**, will travel back into the optical system until they reach the coupler and once again, 10% of the reflected light will be redirected towards the tunable laser and 90% will proceed to the photodetector. The combination of the light sources reflected by the two reflective surfaces (i.e. the fixed fibre tip and the moving membrane) will create an interference pattern which will affect the intensity of the light incident on the photodetector. This device, which has its gain set to 40 dB for our aging study, will convert the photonic energy of the light to an oscillating electrical signal that is sent in our lock-in amplifier while passing through a *Thorlabs EF500* DC block to ensure we are not overloading the input channels of our device.

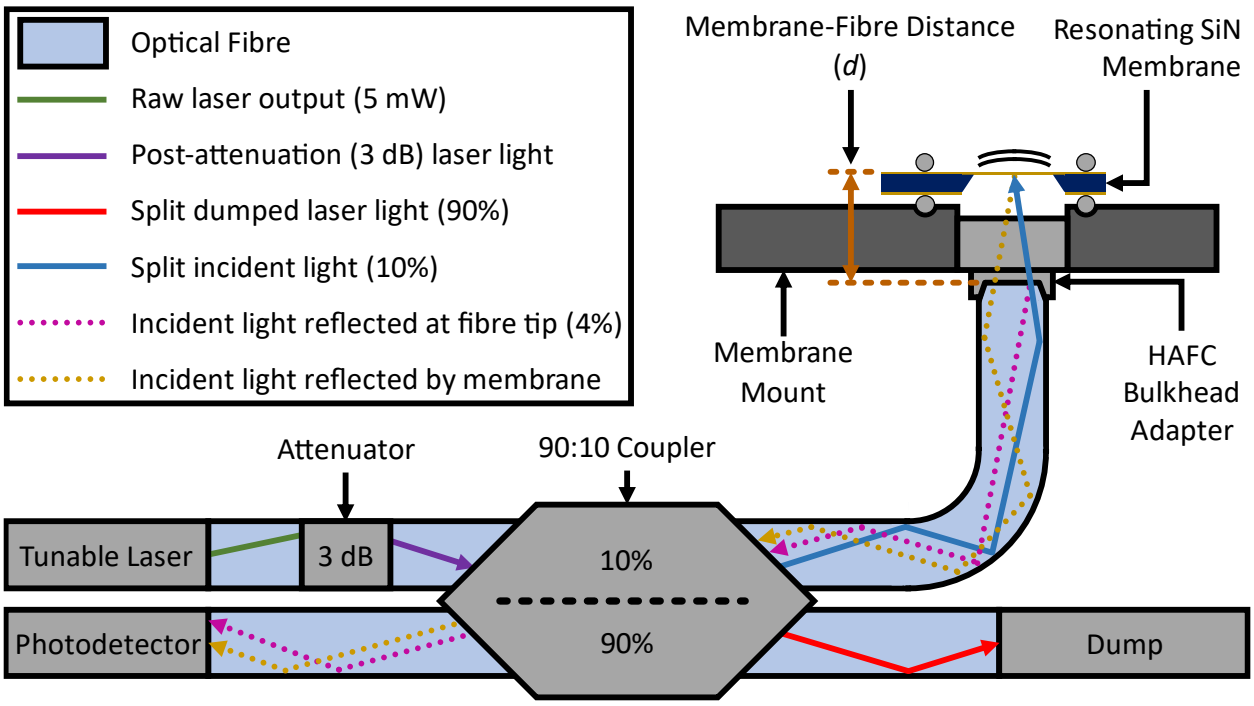


Figure 24: Schematics of simplified laser optical route.

The voltage output of the photodetector (V_{out}) is given by:

$$V_{out} = \frac{V_{min} + V_{max}}{2} + \left(\frac{V_{min} - V_{max}}{2}\right) \cdot cos\left(\frac{4\pi d}{\lambda}\right), \quad (14)$$

where λ is the wavelength of the laser in nm, d is the distance between the fibre tip and the moving membrane in nm. V_{min} and V_{max} are the minimal and maximal achievable voltages which will respectively correspond to destructive and constructive interferences between the two reflected lights. By sweeping the wavelength of the laser or the membrane-fibre distance, we can plot the voltage of the signal and obtain the fringe representing the interference pattern as presented in **Figure 25**. Since the gross distance between our membrane and the fibre tip is fixed by the position of the holder, we rely on tuning the wavelength of the laser to coarsely change the voltage of our signal until it falls into the centre of the quasi-linear region. It is important to be in the centre of this region since it is where the sensitivity of the photodetector to the variation of light intensity (noted by the slope) is maximal, as opposed to the peaks. This allows for the continuous inferred frequency signal to be more precise. During the long-term period of 135 days, the fringe voltage drifted from its parked centre value, by +50 mV. This increment still places the signal well within the quasi-linear region of the fringe and is a testament of our large temperature stability, especially compared to the other setups in our laboratory.

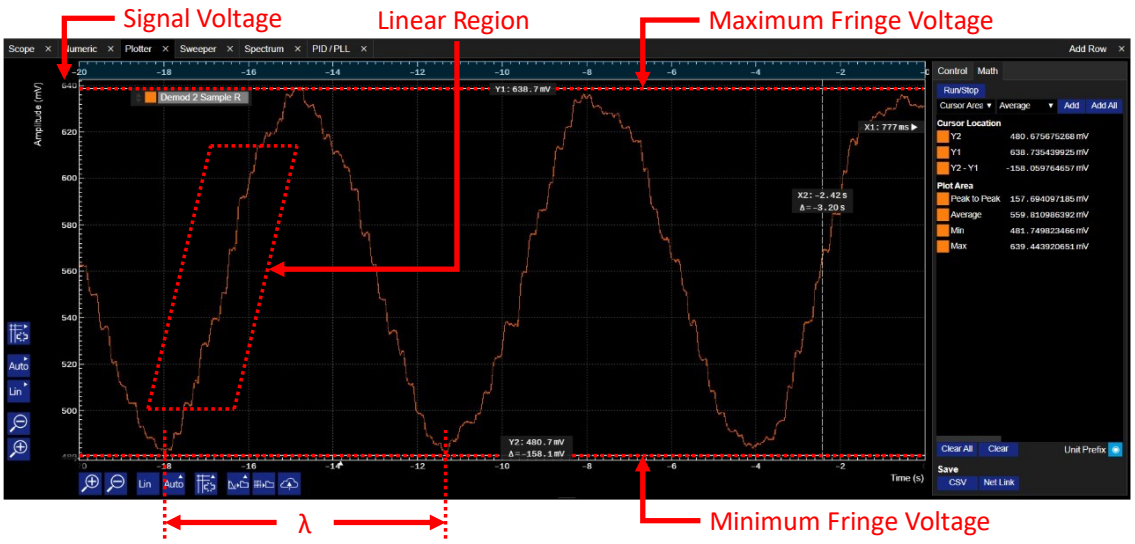


Figure 25: Fringe of the laser's signal.

3.3.2 Lock-In Amplifier Frequency Measurement

The frequency of the membrane can be extracted from an oscillator of known frequency that is synchronized to our resonating MEMS through the manipulations of the time-dependent voltage signal from our oscillating device. The real-time voltage converted from the light intensity is captured by the photodetector and sent to our lock-in amplifier where it will be demodulated. Using the instrument's function, the demodulated voltage in the time domain undergoes a discrete Fourier transform (DFT), through a fast Fourier transform algorithm (FFT), to be transferred to the frequency domain. Once we have the voltage signal in the frequency domain, as represented in **Figure 26**, we excite the frequency corresponding to our desired modes for our membrane (fundamental mode 1-1 and mode 1-2) by sending a voltage output to the piezoelectric actuator on the mount assembly (respectively 7.5 mV and 90 mV). When the voltage amplitude of the resonance is large enough, as observed with the sharp peaks in **Figure 26**, the membrane's resonance frequency can be measured. The distance between the membrane and the tip of the fibre varies as our device vibrates, which causes the interference pattern (from the reflected light sources identified in section 3.3.1) to change. The varying voltage of the signal will affect the extracted

frequency of the membrane that is periodically oscillating. In order to quantify the aging of the device, we must continuously track this changing frequency which is achieved by using a phase-locked loop control system (PLL) that is carefully tuned with the parameters and technique documented in appendix C.

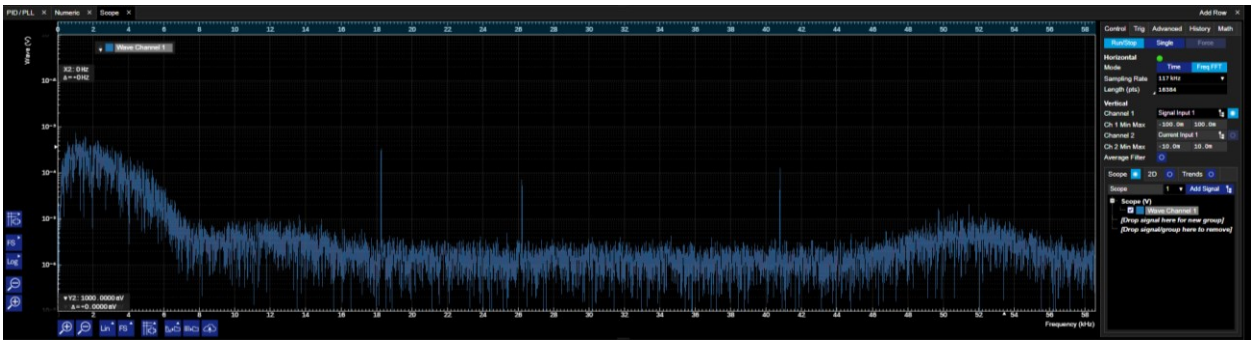


Figure 26: Voltage of the laser's signal in the frequency domain.

PLL is a negative feedback closed-loop control system that can be used to synchronize the frequency of two periodic signals [44]. This control system can be used for many different applications, especially for the frequency tracking of resonators as shown in **Figure 27**. The signal of our resonating device, alongside a reference signal from a controlled crystal resonator internal to the lock-in amplifier, are introduced into a phase detector module that will compute the phase variations between these signals, labelled as the phase error. The phase error is fed back into PID controller, which applies proportional, integral and derivative operations to compute new control parameters that are affected to the internal oscillator [44]. This will result in tuning the frequency of the reference oscillator and ensuring that the phase error between both signals is minimized. When the phase error is close to null, the phases are locked in relation to one another, and their variations are synchronized. Thus, the resonator's frequency is constantly tracked and can be extracted by reading the lock-in amplifier's internal oscillator frequency, which is displayed by the instrument's software.

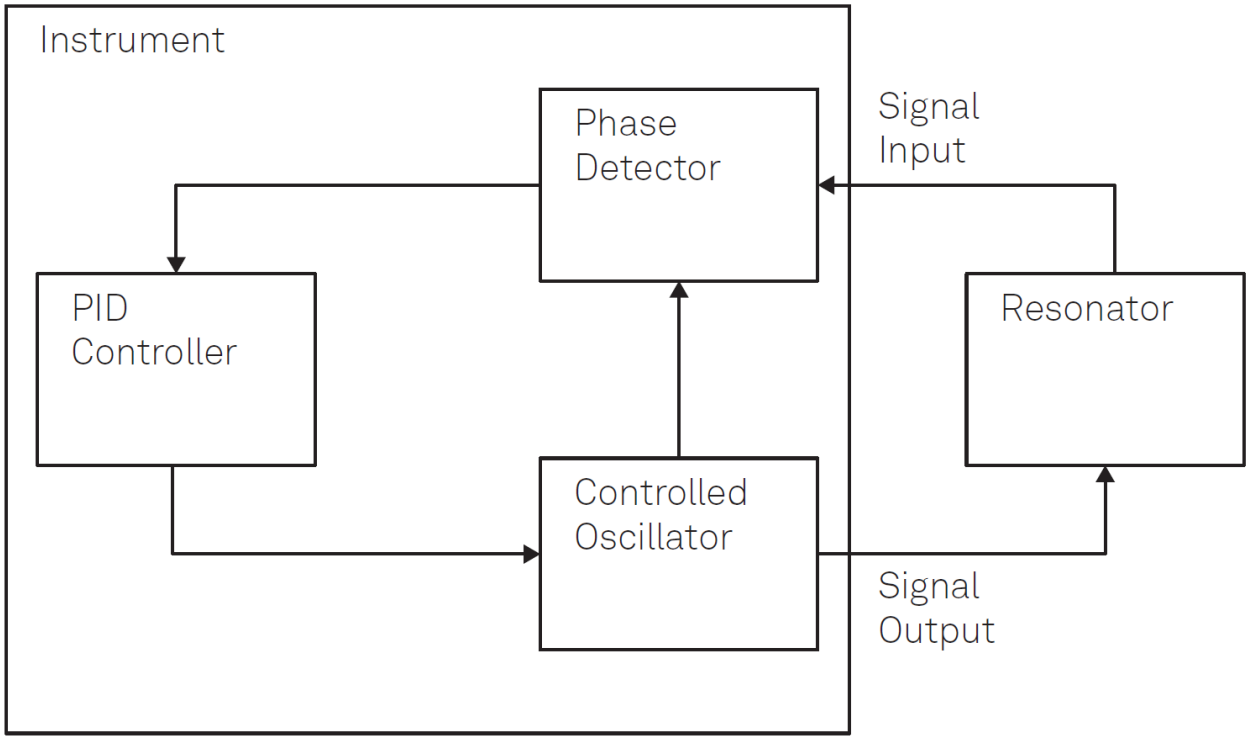


Figure 27: Schematic of a PLL control system used to track the frequency of a resonator. Taken from [44].

3.3.3 Temperature Measurement System

The electrical system used to measure temperatures is much simpler than the optical system used to infer the frequency of our membrane. It relies on a calibrated multimeter (*Keithley DMM6500*) to measure the resistance of a calibrated thermistor which is used to quantify the temperature in the vacuum environment. Additionally, the multimeter measures the resistance of a resistance temperature detector (RTD) which is used to quantify the ambient temperature of the lab. The thermistor (*Measurement Specialties 55036 Glass NTC*) was chosen by the metrologists at the NRC for its excellent long-term stability, while the selected RTD (*TE Connectivity Measurement Specialties NB-PTCO-186*) did not have to be as performant.

The calibrated thermistor, as seen in **Figure 28**, is suspended in the vacuum cell away from the mounting nipple's inner wall to be evenly coupled to chamber by radiation. 4-wire resistance is measured to quantify the temperature in our controlled environment. On the other hand, the

resistance temperature detector (RTD) is held in ambient air by alligator wires, as seen in **Figure 21**, and 2-wire resistance is measured to quantify the temperature of the laboratory.

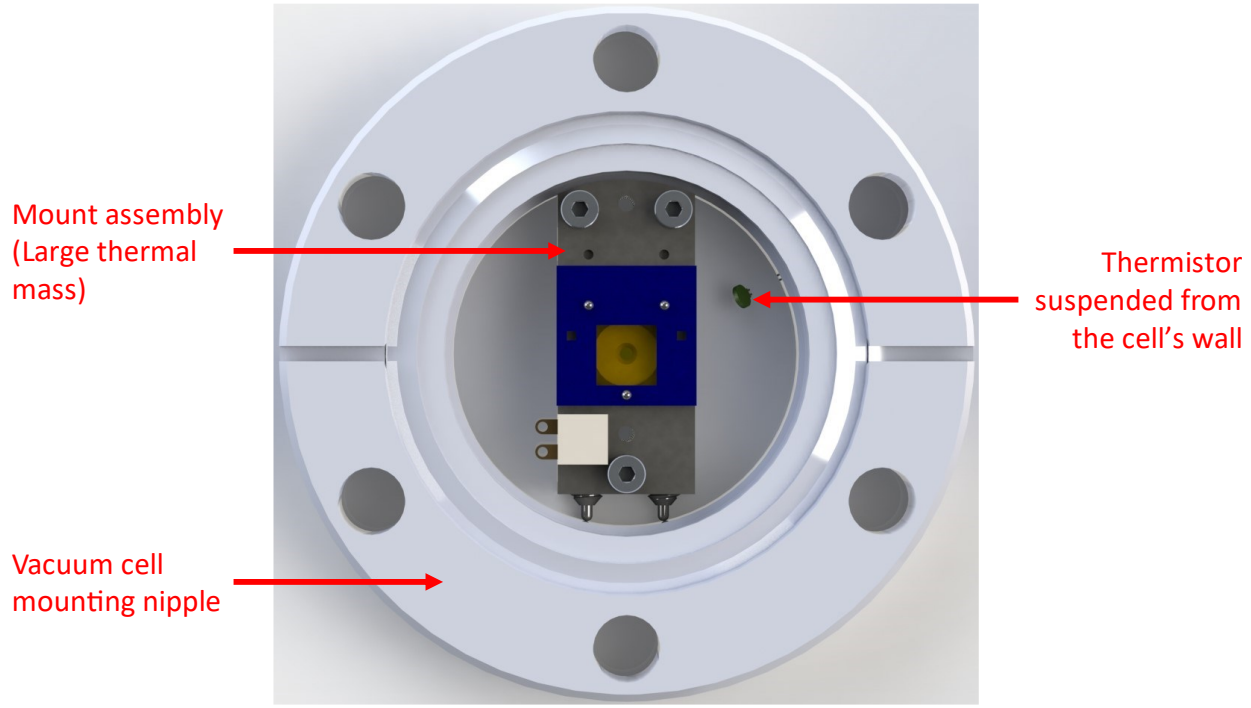


Figure 28: Open view of the vacuum cell with suspended thermistor and mount assembly.

The temperature measurements inside the vacuum cells are essential as they are used to compensate our highly-thermally sensitive frequency measurements and ensure that the recorded drift excludes, as much as possible, factors that are external to the aging of our MEMS. On the other hand, the RTD measurements do not provide specific insight on the occurring aging, but they give very insightful information on how our resonator, even in a highly controlled environment, is largely impacted by changes of temperature inside the laboratory's environment.

3.3.3.1 Resistance Measurement Principles and Methods

The principle of resistance measurements in digital multimeters is shown in the schematic below and relies on the use of constant current. The sensing components, thermistor and RTD in our case, are connected to the multimeter by leads, which will be considered as a device under test

(DUT). Theoretically, the multimeter injects a constant known current through the circuit and measures the voltage drop across the components, which are used to compute the resistance value through Ohm's law.

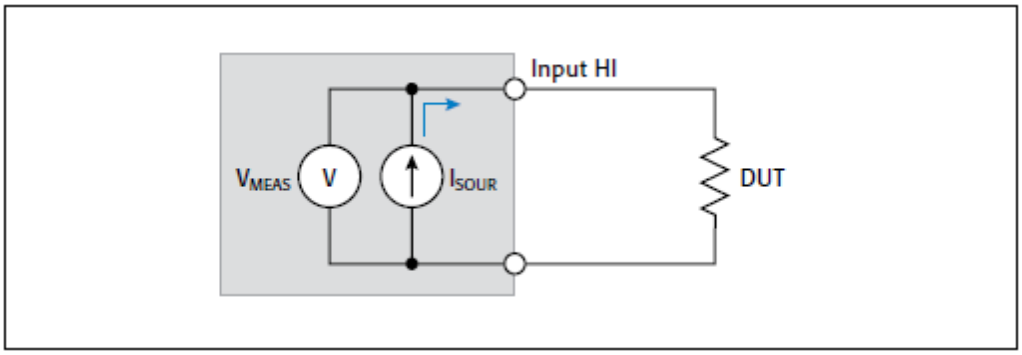


Figure 29: Constant current method resistance measurement principle. Taken from [45].

The main advantage provided by 4-wire resistance over 2-wire resistance is that it can calculate the resistance of our sensing device with more accuracy, especially for low resistance values, since the connection resistance is eliminated. The higher accuracy is the reason why the robust 4-wire method was used for the important thermistor temperature measurements. The different circuitry of both methods is shown in **Figure 30** and **Figure 31**. In the 2-wire method, the resistance of the leads is included in the voltage measured by the multimeter, which is an error source that will result in a measured resistance that is different from the true resistance of the sensing component. These unwanted but inevitable lead resistances are due to various relays, wires, or circuitry that the electrical signal must pass through in the device. By adding an additional pair of lead, as the 4-wire method does, the circuitry now has a pair of wires dedicated to carry the constant current (test leads) where lead resistance losses occur, while the second pair of lead (sensing lead) is directly connected from unknown resistance to the voltmeter. Due to the high impedance of the multimeter's internal voltmeter and the very-small current it will circulate in the

sensing leads loop, very minimal losses will occur, meaning that the measured resistance will be closer to the true value.

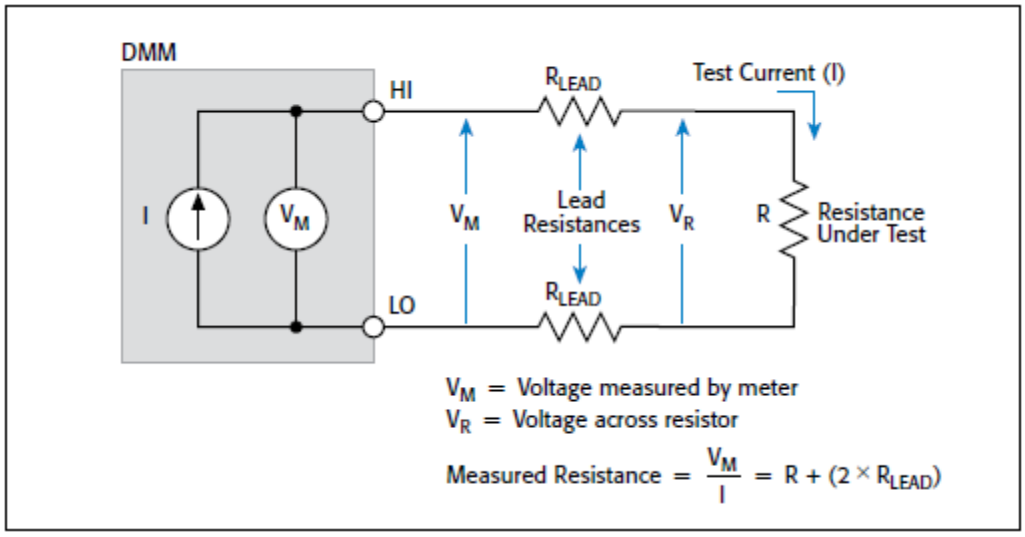


Figure 30: 2-wire resistance measurement principle. Taken from [45].

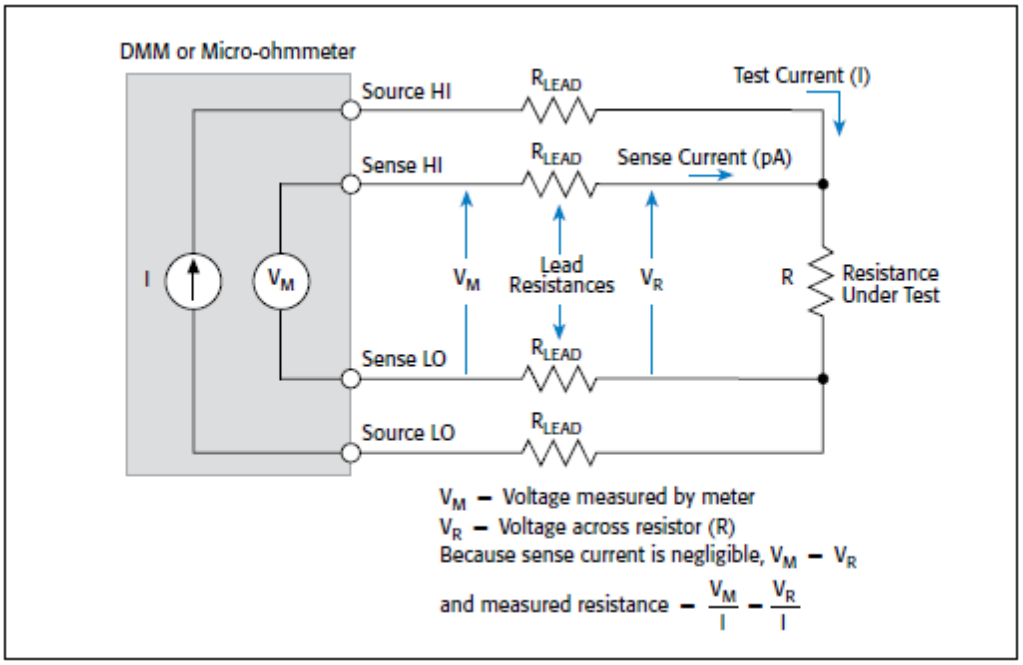


Figure 31: 4-wire resistance measurement principle. Taken from [45].

The resistance range of the multimeter, which dictates the magnitude of the constant current injected in both methods, was fine-tuned for our sensing devices in order to minimize possible self-heating of the components which would affect the temperature in their environment, especially

for the thermistor suspended inside the controlled vacuum setting. On the other hand, lowering the test current also reduces the possible achievable resolution, which is a factor that was considered when selecting the resistance range. For our thermistor, a resistance range of 100 kΩ was selected, which corresponds to a 10 μA test current and a measurement resolution of 100 mΩ. For the RTD, the selected range was of 10 kΩ, which corresponds to a test current of 100 μA and a measurement resolution of 10 mΩ. For the thermistor, a range of 10 kΩ was initially chosen in preliminary tests but was then changed to 100 kΩ since we noticed that the higher current at the smaller range lead to self-heating of the thermistor. On the other hand, there was no problem with the higher current for the RTD which is in air rather than in vacuum.

3.3.3.2 Resistance to Temperature Conversion

With the resistance of the thermistor and the RTD measured by the multimeter, mathematical functions must be used to convert the measured resistances into temperature values. The Steinhart-Hart equation used to convert the thermistor's measured 4-wire resistance (Ω_{Th} in Ω) and obtain the temperature in the vacuum and thermally controlled environment (T_{Cell} in K) is given by:

$$T_{Cell} = \frac{T_{Ref}\beta}{\beta + T_{Ref} \cdot \ln\left({\Omega_{Th}}/{\Omega_{Ref}}\right)}, \tag{15}$$

where T_{Ref} (in K) and Ω_{Ref} (in Ω) are a reference temperature and its respective resistance for the specific device and β (in K) is the temperature coefficient of the specific thermistor. In the case of our specific thermistor, $T_{Ref} = 298.15$ K, $\Omega_{Ref} = 10^4\,\Omega$ and $\beta = 3978$ K. To convert the measured resistance of the RTD (Ω_{RTD} in Ω) into the ambient temperature of the lab (T_{Lab} in °C), the quadratic formula is applied to the equation supplied by the component's manufacturer resulting in the final equation:

$$T_{Lab} = \frac{-a + \sqrt{a^2 - 4 \cdot b \left(1 - {}^{\Omega_{RTD}}/_{\Omega_N}\right)}}{2 \cdot b}, \tag{16}$$

where a and b are coefficients proper to the RTD and Ω_N (in Ω) is the nominal resistance of the RTD at the reference temperature of 276.15 K. In the case of this specific RTD, $a = 3.9083 \times 10^{-3}$, $b = -5.7750 \times 10^{-7}$ and $\Omega_N = 100.12\ \Omega$.

3.4 Data Acquisition and Interpretation

With the proper equipment setup for measurements and the fundamental mode of the membrane excited, Python scripts are used to record the gathered data, save it and execute many signal processing manipulations to convert it from raw data to interpretable results.

3.4.1 Data Continuous Recording

The first Python script, *Continuous Recording.py* which is present in appendix A, is used to connect to the *Keithley DMM6500* multimeter and the *Zurich MFLI* lock-in amplifier through the local area network (LAN) and continuously gather data from these instruments. The script constantly runs throughout the entire recording period or until measurements are stopped. The recorded data is stored in arrays and then saved at intervals of 6 hours into three different text files, with an iterating index, to avoid substantial data losses if an error occurs. **Figure 32** is a schematic representing the data recorded from each instrument.

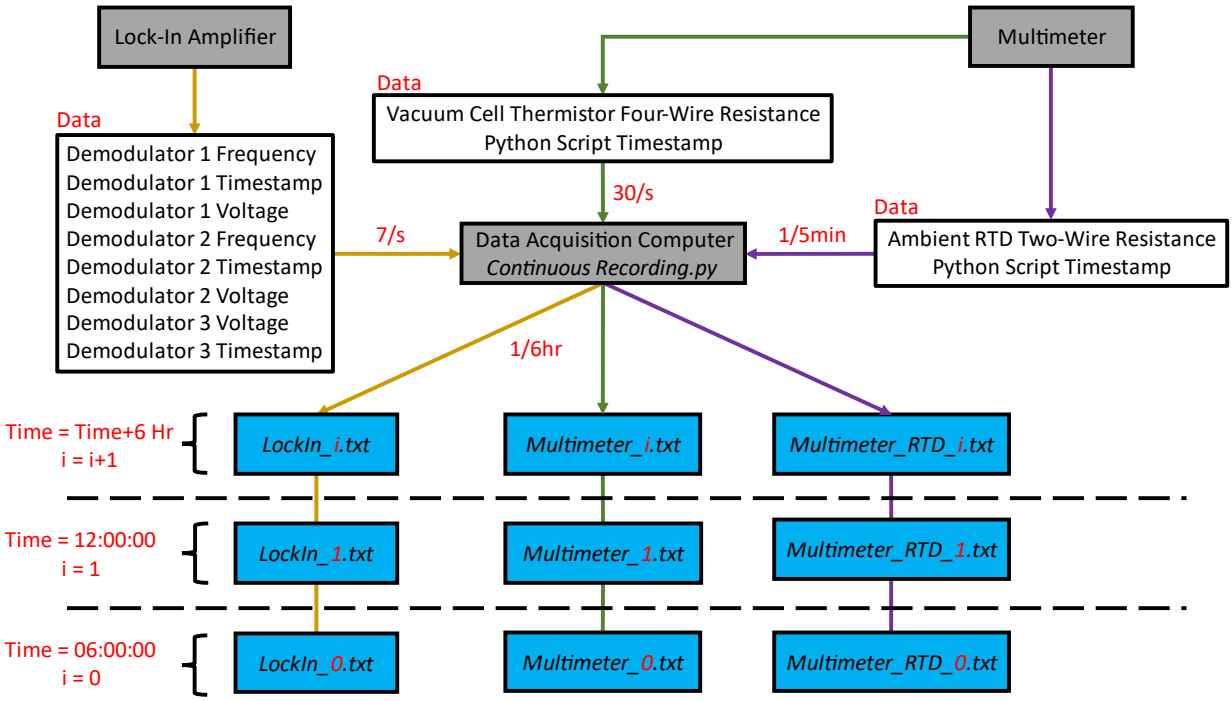


Figure 32: Schematic of recorded data.

The text files containing the data are called datafiles and each have an index *i* to identify the order in which they were recorded. The lock-in amplifier's data is gathered at a sampling rate of 7 samples per second and is stored in *LockIn_i.txt*. The lock-in amplifier's recorded data includes the following: demodulator 1 frequency (corresponds to the mode 1-1 of our vibrating membrane) and time, demodulator 2 frequency (corresponds to the mode 1-2 of our vibrating membrane) and time and demodulator 3 voltage (corresponds to the raw signal of the photodetector) and time. The timestamp used for the lock-in amplifier's data is directly taken from the instrument's clock. The multimeter records the 4-wire resistance of the thermistor inside the vacuum cell at a rate of 30 samples per second alongside the timestamp of the Python script. The thermistor's data is recorded in the *Multimeter_i.txt* datafile. On the other hand, the RTD's 2-wire resistance is recorded once every 5 minutes alongside the Python's script timestamp inside the *Multimeter_RTD_i.txt* datafile.

3.4.2 Data Signal Processing

In order to be manipulated with ease and interpreted correctly, the recorded data stored over multiple text files will have to be merged and decimated (to reduce the excessive amount of data points, in the order of 32,000,000 for raw data over 135 days). This is done by the second Python script, found in appendix A: *Merging and Filtering.py*. This script, whose basic functions are represented in **Figure 33**, will merge all the *LockIn_i.txt* datafiles by appending it to a single array. The same process is done for the *Multimeter_i.txt* and *Multimeter_RTD_i.txt* data, while also converting the recorded resistances to temperature with equations (15) and (16). The merged lock-in amplifier data and the merged multimeter thermistor data are crucial for our results but contain far too many samples to be efficiently manipulated and analyzed. Thus, these merged datasets undergo a decimation process, which will cleanly individually downsample them to a final sampling rate of 3 Hz, by applying a custom Butterworth filter. The decimated data is then sent to a plotting graphical user interface (GUI) shown in **Figure 34**, which allows us to quickly plot the different signals and ensure the data is well recorded with no abnormalities. The plotting GUI function also allows us to apply a rolling median or even trim the datasets if necessary. When the GUI is closed, the merged decimated lock-in and multimeter thermistor data will be concatenated into a single merged dataset, which will retain all their signals as presented in **Figure 33**, but will have one global interpolated timestamp. The merged data will be saved in a text file labelled *Merged_Filtered_Data.txt*.

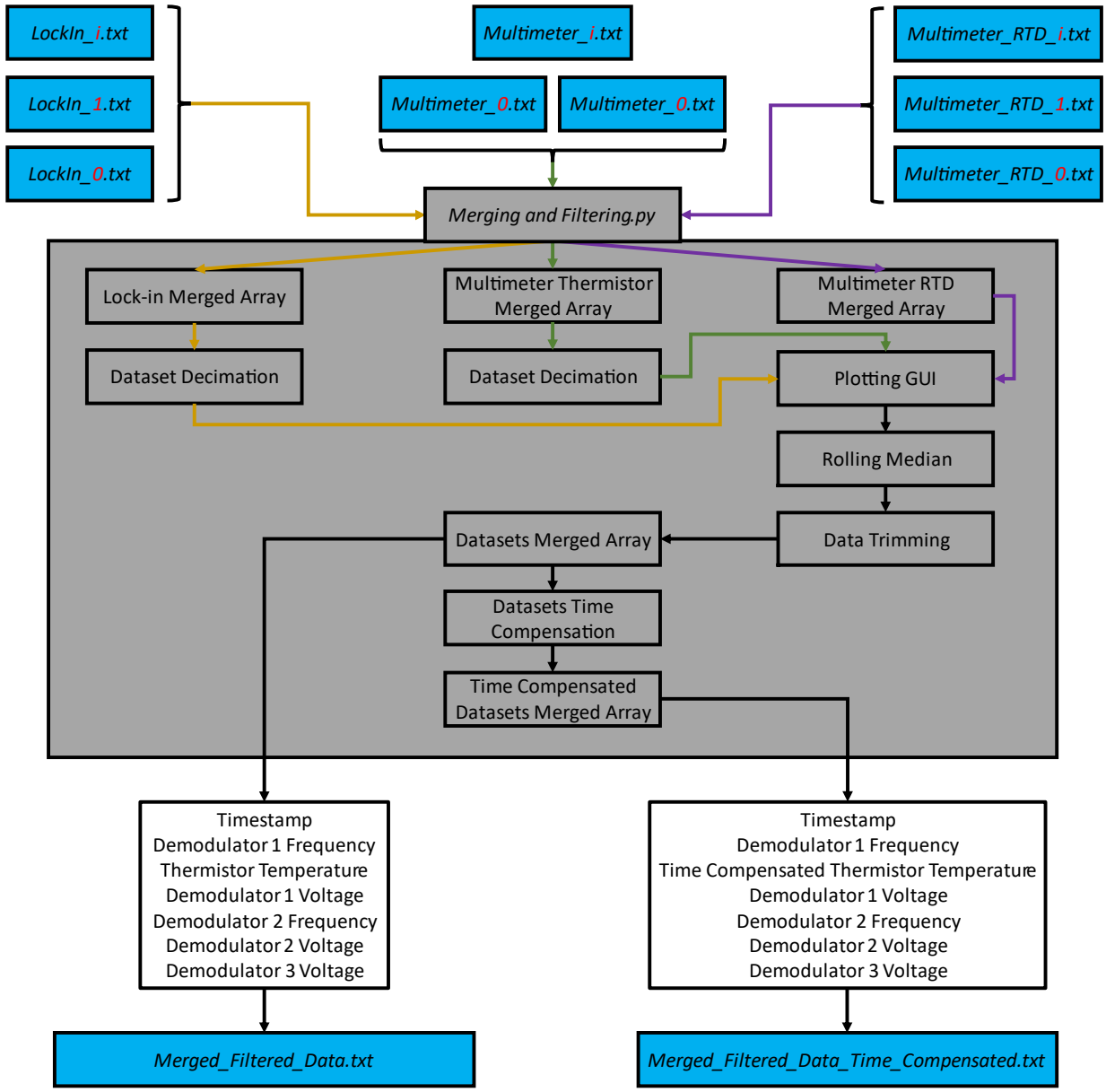


Figure 33: *Merging and Filtering.py* script basic functions.

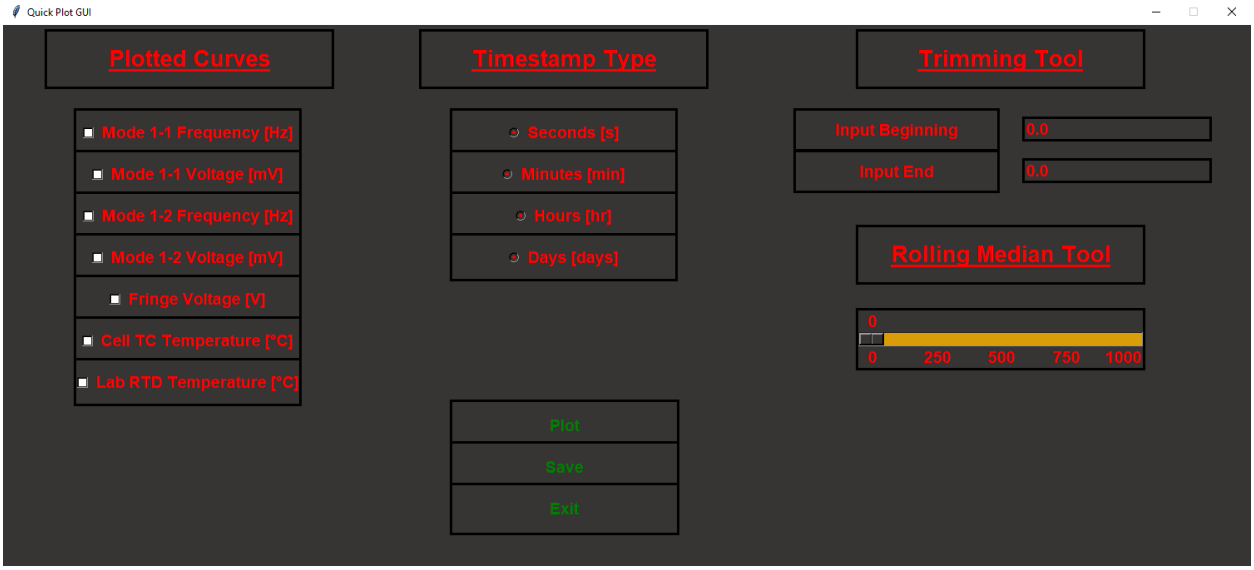


Figure 34: *Merging and Filtering.py* plotting GUI.

The *Merging and Filtering.py* script also features a time compensation algorithm which will find the lag between the peaks in the thermistor's temperature data and the peaks in the lock-in frequency shifts to time-compensate the temperature data accordingly. The lag comes from the changes of temperatures which are almost instantaneously measured by the thermistor but take substantial time to affect the membrane's frequency in reason of the holder's large thermal mass. Thus, a fine-tuned peak finder algorithm is applied to the fundamental mode resonance frequency signal and the thermistor temperature signal. Peaks are identified by sifting through all the points of the signals and highlighting the ones that satisfy established criteria, which are represented in **Figure 35**. To be a valid peak, the identified summit must meet a certain criterion of height, distance to another potential peak and prominence. The height criterion is described as a minimum value for the analyzed signal to be qualified as a peak. The distance criterion is the minimum time difference required between to respective summits of the same signal to be considered as a peak. The prominence is the minimum value difference required between a summit and its local minima to be flagged as a peak. This algorithm will match corresponding satisfying peaks between the two

signals. The script will then compute the mean time delay between the membrane resonance frequency and the thermistor's temperature signal, by averaging all established peak time lags. The thermistor's temperature data will be re-interpolated by incorporating the calculated mean time lag into its timestamp. As a result, the thermistor's data will be compensated for the existing time lead it had over the frequency signal. The time compensated merged data is saved under another text file labelled *Merged_Filtered_Data_Time_Compensated.txt* and contains the same signals as the *Merged_Filtered_Data.txt* file, with the exception of having an interpolated and time compensated version of the thermistor's temperature data.

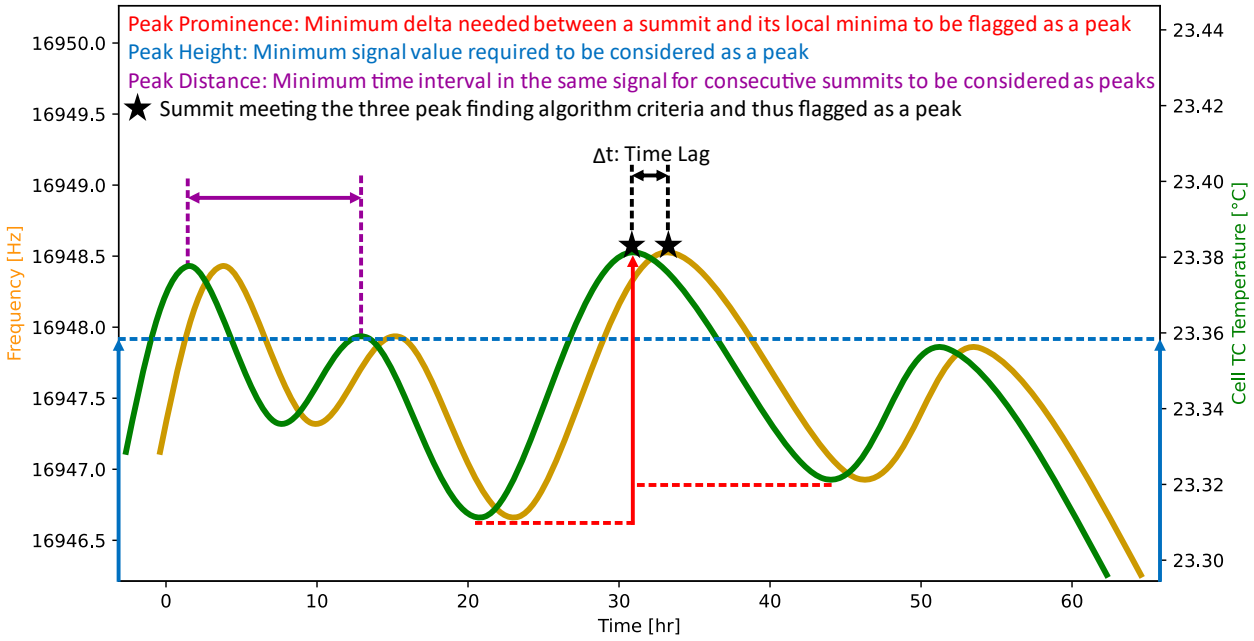


Figure 35: *Merging and Filtering.py* peak finding algorithm criteria.

3.4.3 Data Interpretation

With the data recorded and the signal processing successfully applied, we can now use the result generating scripts *Colour Map Plotter.py* and *Frequency Drift Plotter.py*, included in appendix A, to gather all of the necessary plots and values. The *Colour Map Plotter.py* script is very straightforward and essentially loads the different

Merged_Filtered_Data_Time_Compensated.txt datasets, appends them into one large array and then uses the contained data to generate the following result plots:

- Membrane frequencies and vacuum cell temperature as functions of time,
- Fundamental mode frequency as a function of vacuum cell temperature with time dependency colour map.

The *Frequency Drift Plotter.py* script will load the different *Merged_Filtered_Data_Time_Compensated.txt* datasets but will conduct many more mathematical operations to generate results. The main purpose of this script is to extract the true frequency drift in our results, which requires a temperature compensation of our data. To do so, the total data is sectioned in slices of 6 hours to fit and extract a linear trend for the frequency of the membrane as a function of the temperature of the cell. We can then compute the mean of the offset and the slopes of the equations issued from the linear trends to form a global linear equation that allows us to predict the membrane's resonance frequency from the cell's temperature. To gather our aging frequency drift, we take our membrane frequency as a function of time for the whole recording period of 135 days and subtract it by the predicted thermal induced frequency drift that depends on the recorded cell temperature at the respective timestamp. The initial frequency of the membrane is also subtracted from the frequency differential. Finally, this total differential frequency is divided by the initial frequency of our measurements to obtain the frequency drift as a function of time. Compensating the membrane frequency's data for the temperature changes ensures that the drift evaluated is linked solely to aging. At this point, using educated guesses, the script can fit the drift data using either a double logarithmic model or hybrid logarithmic-linear model.

4. Results

4.1 Aging Study Findings

The results obtained for our membrane represent the aging that it has undergone over a period of 135 continuous days. The first interpreted results, after applying the signal processing method explained in section 3.4.2, are the raw data signals illustrated in **Figure 36**. They include the membrane's resonance frequency for its mode 1-1 (in black) and mode 1-2 (in cyan) as well as the temperature of the thermistor inside the vacuum cell (in green), as functions of time.

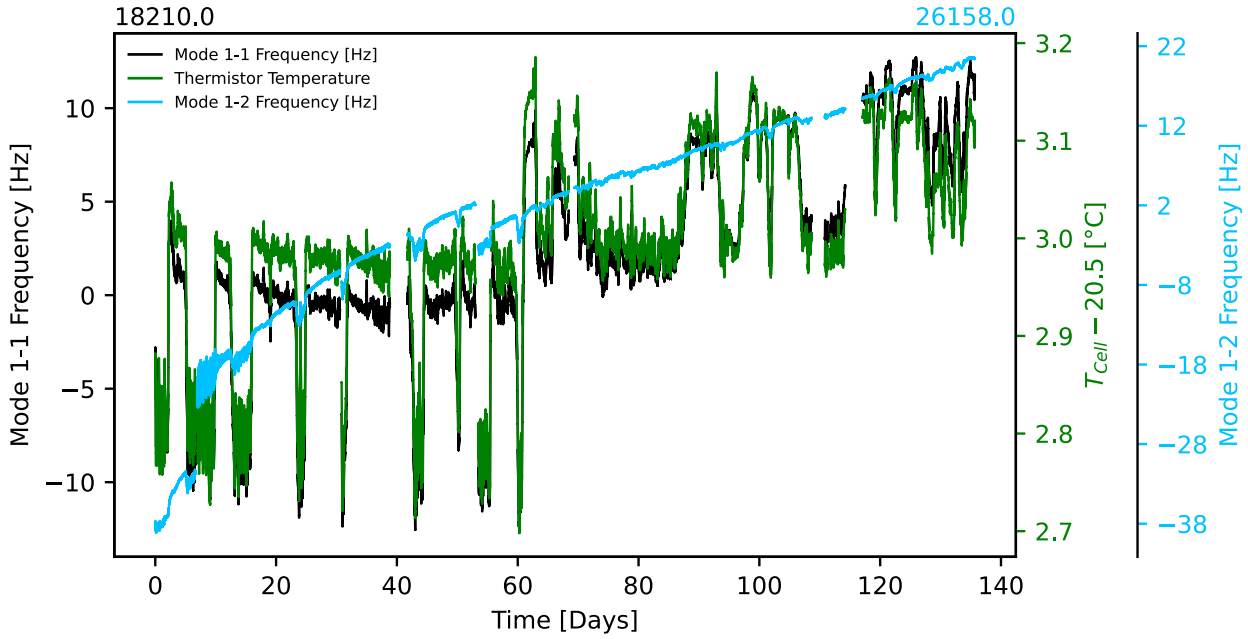


Figure 36: Raw data results: membrane mode 1-1 and mode 1-2 resonance frequency and vacuum cell temperature as functions of time.

The resonance mode that we use to study the membrane's aging is the mode 1-1 since it is very stable when compared to the degenerate mode 1-2, as seen in **Figure 36**. It is likely that mode 1-2 mixes with mode 2-1 over time, in a variable fashion that causes more drift. The same possibility does not exist for non-degenerate mode 1-1. Since the start of the measurements, the frequency of mode 1-2 continuously drifts in an upward fashion with an overall frequency

differential of 60 Hz. On the other hand, the bulk of the frequency for the mode 1-1, which appears to be much more stable, fluctuates around a stable central value of 18,210 Hz. Furthermore, the frequency of mode 1-1 follows, with a high fidelity, the temperature changes in the vacuum-cell, as displayed by the thermistor's curve. The close correlation between the behaviour of the thermistor's temperature, which is in a very stable and controlled environment, and the frequency of the mode 1-1 illustrates how our membranes excel in thermal sensing. A cyclic behaviour is observed in the signal of the thermistor and frequency of mode 1-1, where the data alternates between a lower and higher plateau of values throughout the entire measurement period. This cyclical temperature change was hypothesized to be induced by the heating, ventilation, and air conditioning system (HVAC) of our laboratory. To investigate this behaviour further, we have added a RTD that measures the lab temperature every 5 minutes, outside of the vacuum chamber and the controlled air bath.

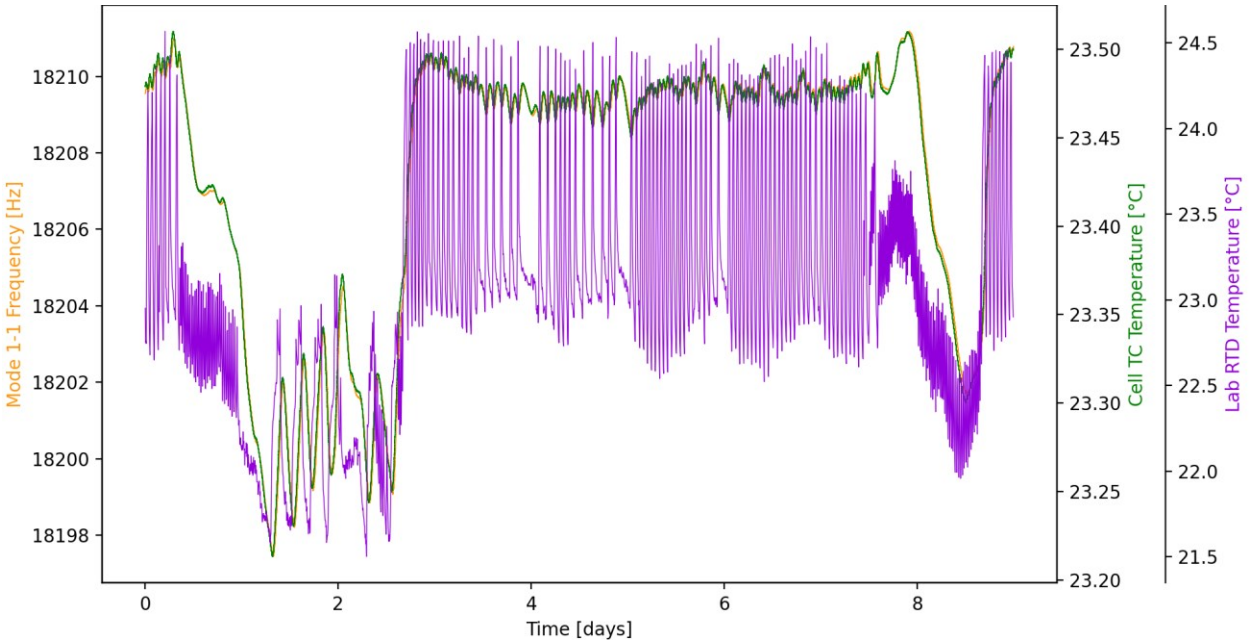


Figure 37: Raw data results: membrane mode 1-1 resonance frequency, vacuum cell temperature and laboratory temperature as functions of time.

Figure 37 depicts the raw data for the frequency of the fundamental mode of our membrane (in orange), alongside the vacuum cell thermistor's temperature (in green) and the newly added laboratory RTD temperature (in purple). As expected, the temperature swings measured in the laboratory dictate the behaviour that was observed in the vacuum cell's data (**Figure 36**) with a delay consistent with the thermal mass of the vacuum cell. Thus, we concluded that the cyclic ramping up and down of the temperature and frequency was in fact due to the HVAC system and was not an anomaly due to an instability in the measurement setup.

As we indicated in section 2.5.5, to avoid incorporating the piezoelectric actuator's parasitic heat, the recording of data is started after the time required for the heating of the piezoelectric actuator to reach steady state. This transient heating period can be seen in **Figure 38** below and is on the same order of magnitude as the transient response time of the membrane holder and vacuum chamber.

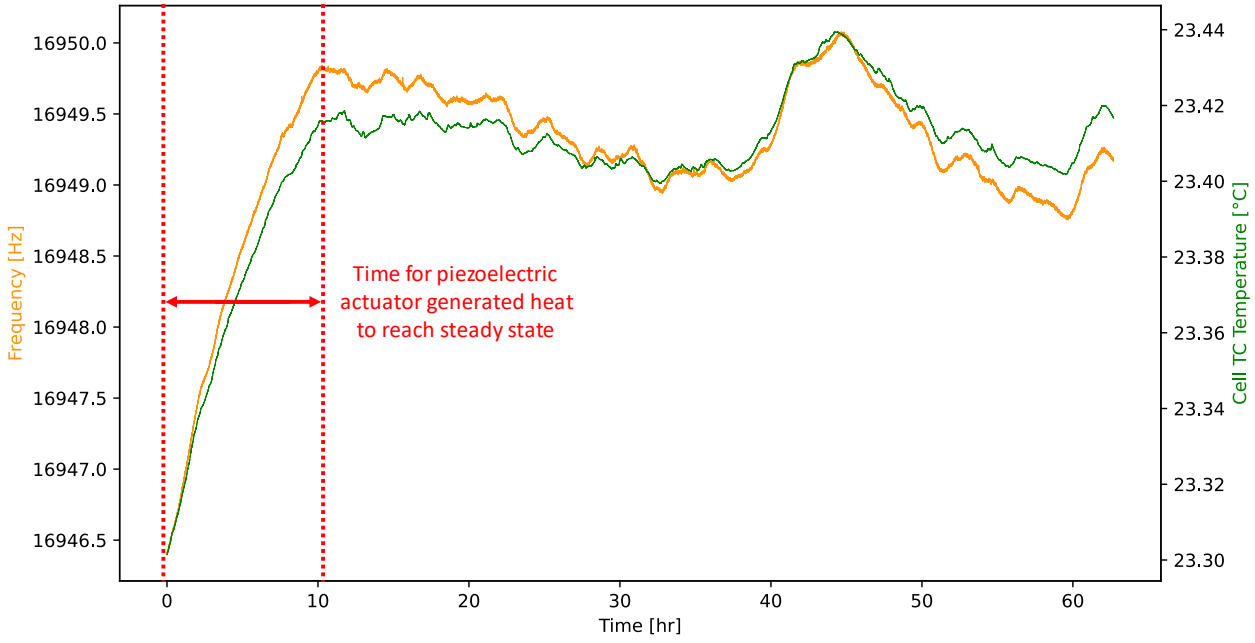


Figure 38: Time required for the piezoelectric actuator's generated heat to reach steady state following the beginning of actuation.

Using the data of **Figure 36**, we were able to produce a colour plot where the frequency of the fundamental mode of our membrane was plotted as a function of the thermistor temperature in the vacuum cell, while a colour gradient indicated the dependency of both variables to time. **Figure 39** is a colour map of the data, from which we infer a global linear trend that we use to predict the frequency shift as a function of temperature and thus compensate our raw frequency as a function of time data. The drift due to aging in the colour map is represented by the gradual vertical displacement of the linear trend (i.e., the dashed green line schematized in **Figure 39**). A problematic artifact in the colour map, that affects the generated linear trend, is the elliptical shapes present at the corners of lines. This type of abnormality is created by the time lag existing between temperature changes measured by the thermistor and by the membrane's fundamental mode's frequency. The thermistor inside the vacuum cell can measure the temperature changes instantly since it is suspended in the vacuum cell, while there is a delay before the membrane's frequency is affected by this thermal change because our device is fixated on an assembly with a large thermal mass as seen in **Figure 28**.

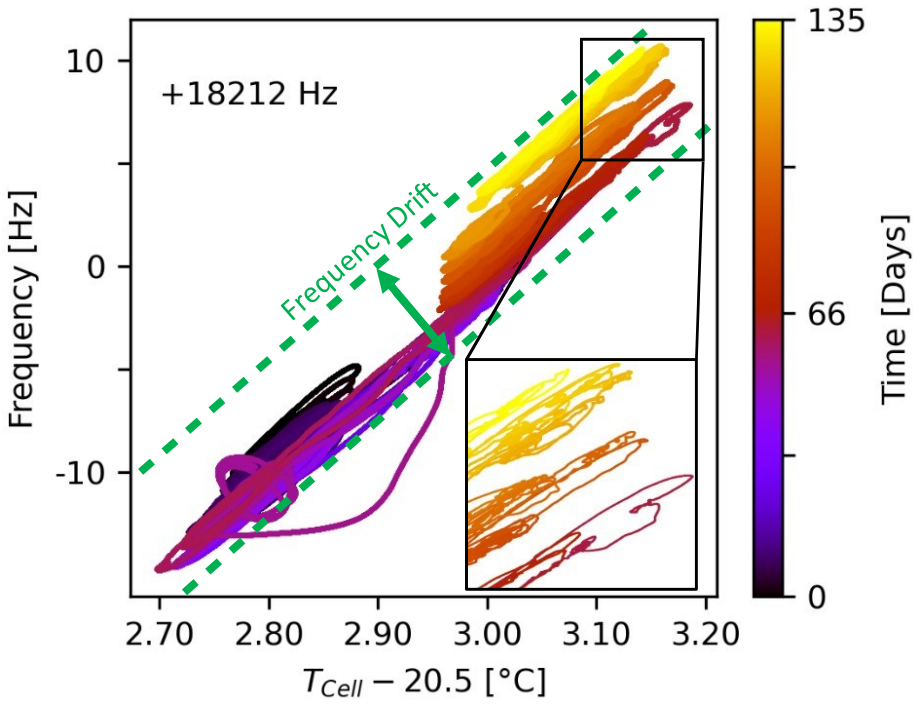


Figure 39: Time dependent colour plot with membrane's fundamental mode frequency as a function of vacuum cell's temperature without time compensation.

Thus, lag compensation for the delay in the measurements of temperature and our membrane frequency, which is explained in section 3.4.2, is essential and necessary to obtain a clear colour plot with which we can obtain better linear trends to extract the frequency drift induced by temperature changes. As we can see in **Figure 40**, the time lag averages around 700-1000 seconds and is compensated accordingly by our *Merging and Filtering.py* script.

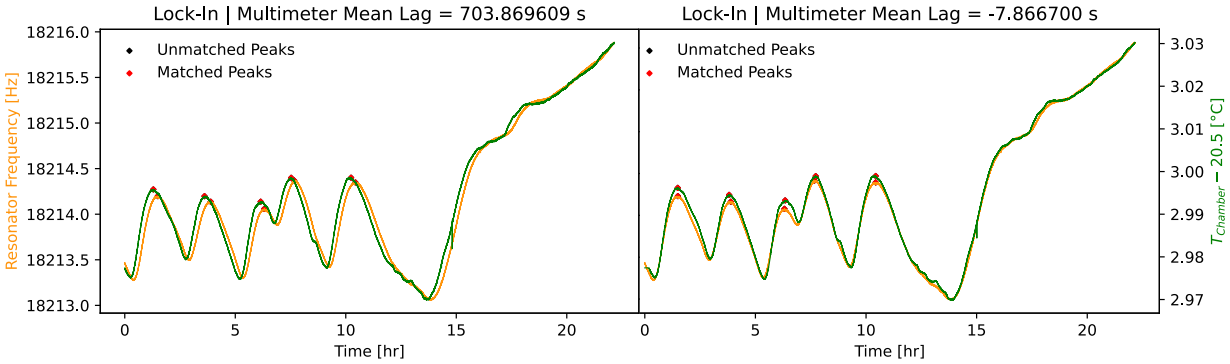


Figure 40: Portion of the signals before lag compensation (left) and after lag compensation (right).

The time lag, which is of a very small value compared to the overall span of the aging study we conducted, might not seem to make a crucial difference, but it significantly affects the analysis and computation of the obtained results. We can see in **Figure 41** that the time compensated colour plot, compared to the non-compensated colour plot depicted in **Figure 39**, produces results that have clearer linear trends while also eliminating elliptical abnormalities due to the time lag. The inferred frequency drift from this plot is around 4 Hz. The linear trend that allows us to compute the temperature dependent frequency drift (f_{Temp} in Hz) as a function of temperature of the vacuum cell (T_{Cell} in °C) is represented by the following equation:

$$f_{Temp} = 43.370931 \cdot T_{Cell} + 18082.177740. \tag{17}$$

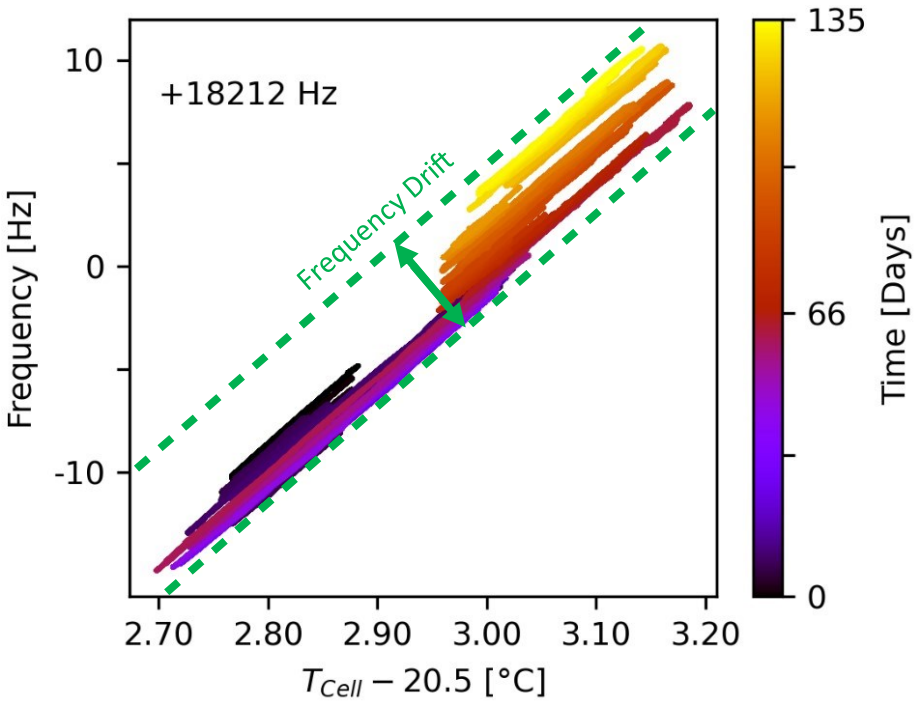


Figure 41: Time dependent colour plot with membrane's fundamental mode frequency as a function of vacuum cell's temperature with time compensation.

With the global linear trend generated from the corrected colour plot, we can now compensate the membrane's frequency shift for its thermal drift component and finally obtain the aging frequency drift plot showcasing the stability of the fundamental mode of our SiN membrane.

In other words, we produce **Figure 42**, which is essentially illustrating the drift of the green dashed line of **Figure 41** as a function of time. As shown in **Figure 42**, the drift presents an irregularity where the obtained frequency drift data oscillated between a regime of upper and lower values. This abnormality occurs since the overall slope of the linear trend computed for the temperature compensation is not perfectly representative of the data and must be fine-tuned. When applying a positive correction factor of 8% to the overall slope of the compensation linear trend, the previously present upper and lower regimes collapse together, forming a smooth and continuous drift line as presented in **Figure 43**. In this case, the new frequency to temperature correlation is:

$$f_{Temp} = 46.840605 \cdot T_{Cell} + 18082.177740. \quad (18)$$

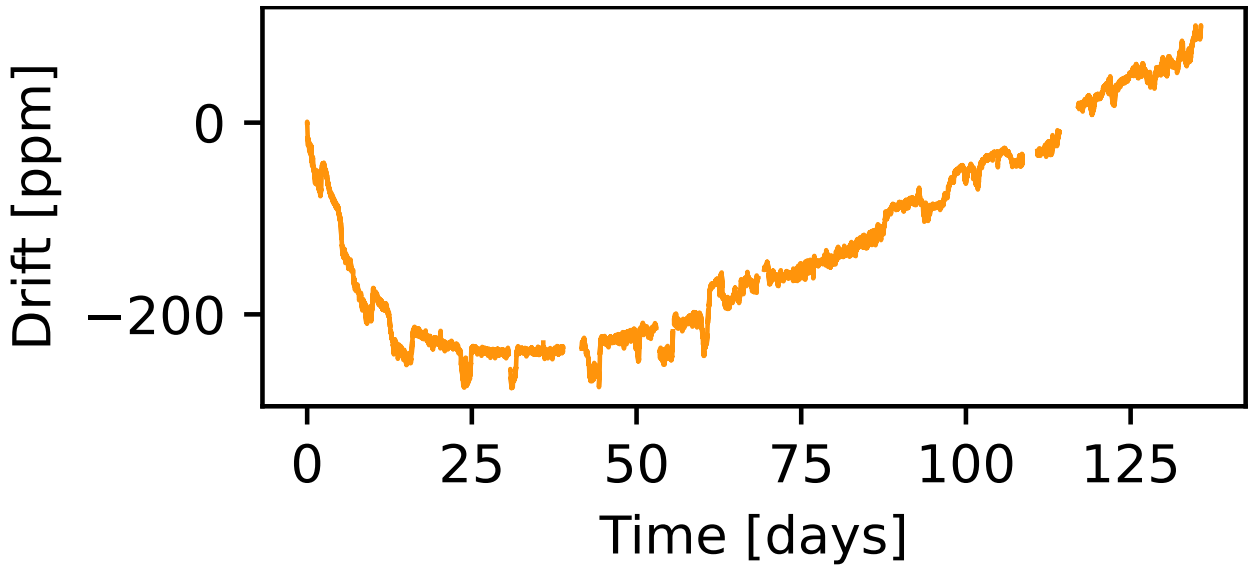


Figure 42: Membrane's fundamental resonance frequency drift without slope fine-tuning.

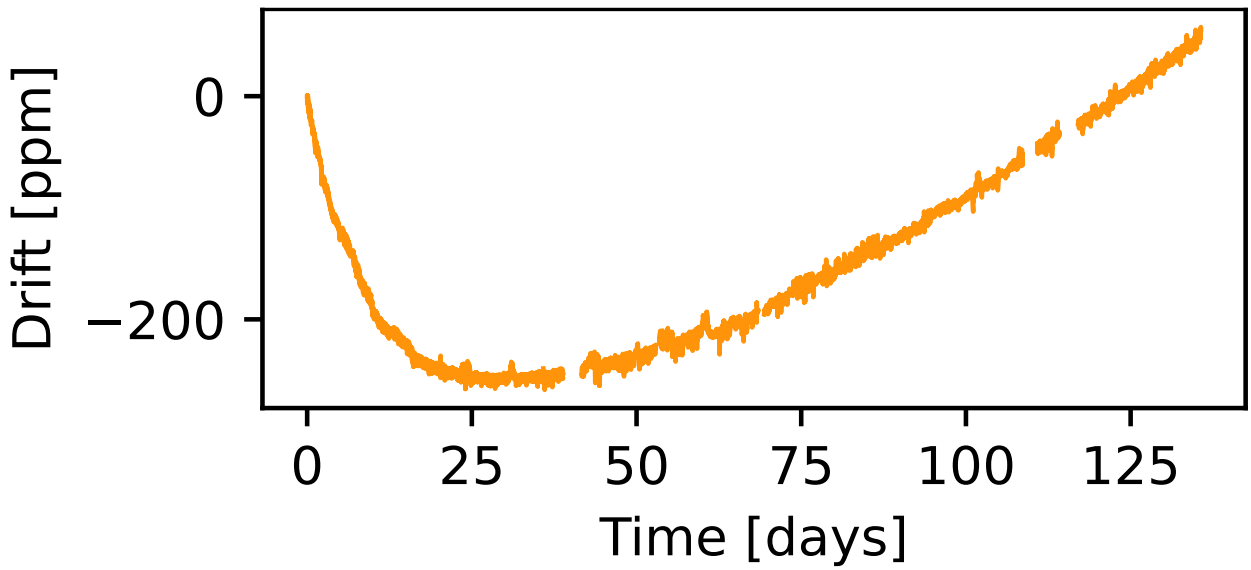


Figure 43: Membrane's fundamental resonance frequency drift with 8% positive slope fine-tuning.

The final drift plot, presented in **Figure 44**, which can finally be generated given the different corrections the data has undergone, allows us to analyze the stability of the fundamental resonance mode (in green) and the mode 1-2 (in cyan) of our membrane. As expected, the stability of the mode 1-1, which reached a value of 300 ppm for 135 days of aging measurement, is inherently larger than the mode 1-2 which is around 1750 ppm for 135 days of aging.

The double logarithmic model that was proposed by several authors, as seen in **Figure 3** of section 2.3.1, is observable in the drift data. Around the timestamp of 30 days on our aging data in **Figure 44**, we can observe the reversal in the aging regime, a topic we touched upon in section 2.3.1. This initial phase where the frequency drifts in a negative fashion is due to the desorption of the water adsorbates. This contamination by adsorption has sub-mechanisms responsible for aging of our membrane, as expanded upon in section 2.5.2 to 2.5.4. The aging frequency drift of our fundamental mode has thus been fitted with a double logarithmic equation, following the proposed models. The obtained double-logarithmic equation depicting the sustained frequency

drift due to aging (f_{Age} in ppm) as a function of time (t_{Age} in s) has the following form and parameters:

$$f_{Age} = -203.51 \cdot \ln\left(3.33\text{x}10^{-2} \cdot t_{Age} + 11794.64\right) + 16463.93 \cdot \ln\left(4.52\text{x}10^{-9} \cdot t_{Age} + 1.12\right). \quad (19)$$

The trend equation is only valid for the measurement period it underwent and would not be accurate enough to forecast expected drift for a period much larger than 135 days. It was also observed that the second logarithmic term's time dependent parameter had a very small value, making it essentially a linear term. We can essentially obtain the same curve with the hybrid logarithmic-linear function:

$$f_{Age} = -198.41 \cdot \ln\left(2.64\text{x}10^{-6} \cdot t_{Age} + 0.89\right) + 6.41\text{x}10^{-5} \cdot t_{Age} - 13.78. \quad (20)$$

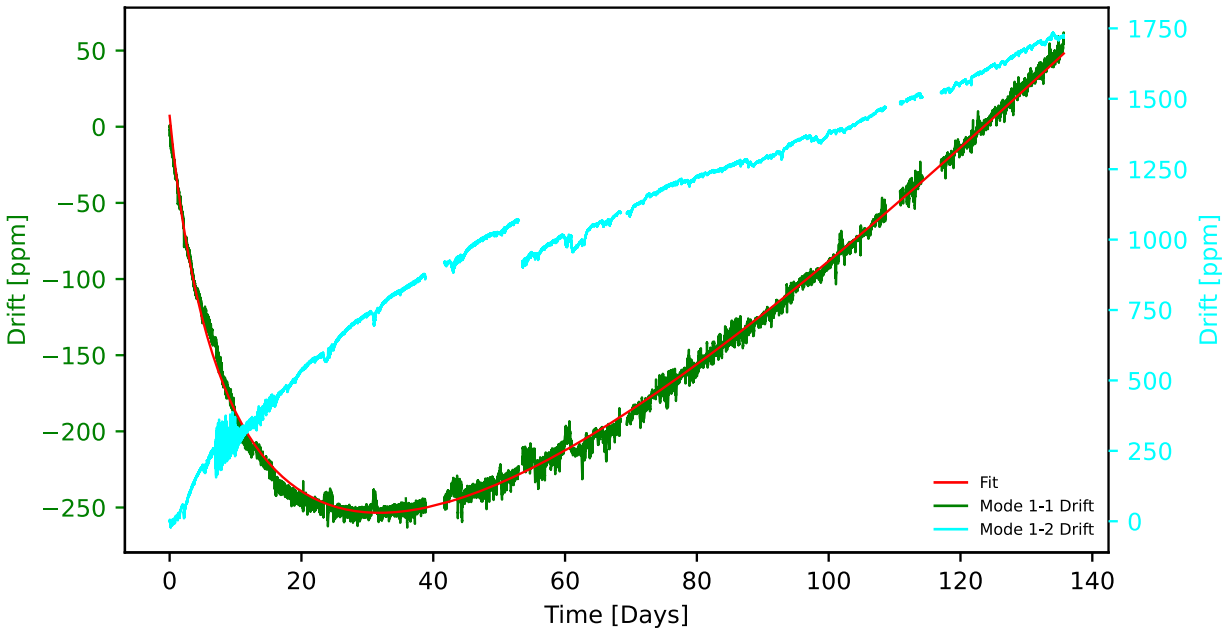


Figure 44: Membrane mode 1-1 (with double-logarithmic fit) and mode 1-2 frequency drift.

The achieved stability of our SiN resonating membrane is of 300 ppm/135 days, which is substantially larger than the metric of 1 ppm/year established by the metrology team NRC, an objective we desired to achieve for the sensor project that we jointly develop. It is also much lager than the typical 0.02 ppm/year achieved in clock resonators. This difference is not surprising since

our device, unlike clocks showcased in the sections 2.3 and 2.4, is not developed with stability as its sole focus. Interestingly, the stability of our membrane, when weighing the drift value by the typical resonator thickness, is within same order of magnitudes as that of clocks, which points to aging being mostly a surface effect.

Even though we have not reached the target wished by our metrology collaboration, the stability study we conducted allows us and other scientists to gain important insight on the stability of silicon nitride through time. The findings we collected could thus allow us to calibrate our devices or even compensate measurements over a longer period than a few days of usage. The obtained frequency drift is not nearly as big as the expected overall drifts computed in sections 2.5.2 to 2.5.4 and presented in **Table 2**. There is a large magnitude difference between the experimental results and overall computed drift tabulated. This difference exists because the bulk of drift due to the aging sources explored in section 2.5 occurs in the first few minutes of the vacuum pump down period, when measurements are not taken and most of the active sites of the membrane become unoccupied. The impossibility to acquire measurements during this critical initial period is due to the fact that:

- While the turbomolecular pump creates the vacuum environment, the resonator's frequency shifts greatly making it practically impossible to track through PLL,
- We must wait at least 10 hours to allow the piezoelectric actuator's generated heat to reach steady state as explained earlier and depicted in **Figure 38**.

4.2 X-ray Photoelectron Spectroscopy Measurements

Thanks to our collaboration with the metrology group at the NRC, we had the possibility to perform X-ray photoelectron spectroscopy (XPS) measurements on our aged membrane and another membrane for reference. XPS is a type of measurement that reveals the elements present

on the surface of a tested object. In our case, we use it to see what type of contaminants are on the surface of our membrane. The obtained photoelectron spectra from the measurements have characteristic energy levels that are specific to particular elements. Thus, with the measured energy levels, we can see which contaminants are present on our membranes. The reference tested sample is a 6 × 6 mm SiN membrane that was stored in a Petri dish in the ambient environment of our laboratory. The other sample is our membrane that was used throughout this aging study. While these samples are different, considering that our aging membrane spent 135 days under vacuum, they both share the same geometry and were manufactured in the same batch of nanofabrication, that ended with KOH etching on May 19, 2022.

4.2.1 Experiment Timeline

Contamination occurs at every instant a device is exposed to an ambient setting. The longer it is stored, the more it will suffer contamination from different elements and compounds. A timeline, as presented in **Figure 45**, is necessary to have a better understanding and view of the journey of the tested samples. The XPS tested samples were both stored in an ambient setting from their creation date (May 19, 2022) to September 12, 2022. At the latter date, the membrane used for aging was placed inside the vacuum environment for 135.75 days, until January 26, 2023. The aging membrane was then stored in ambient setting, just like the reference membrane, until they were both brought to the NRC and put inside the XPS vacuum environment for tests conducted on February 23, 2023. The membranes were then unloaded from the XPS vacuum environment and stored inside a desiccated plexiglass enclosure that is kept at 10 Torr.

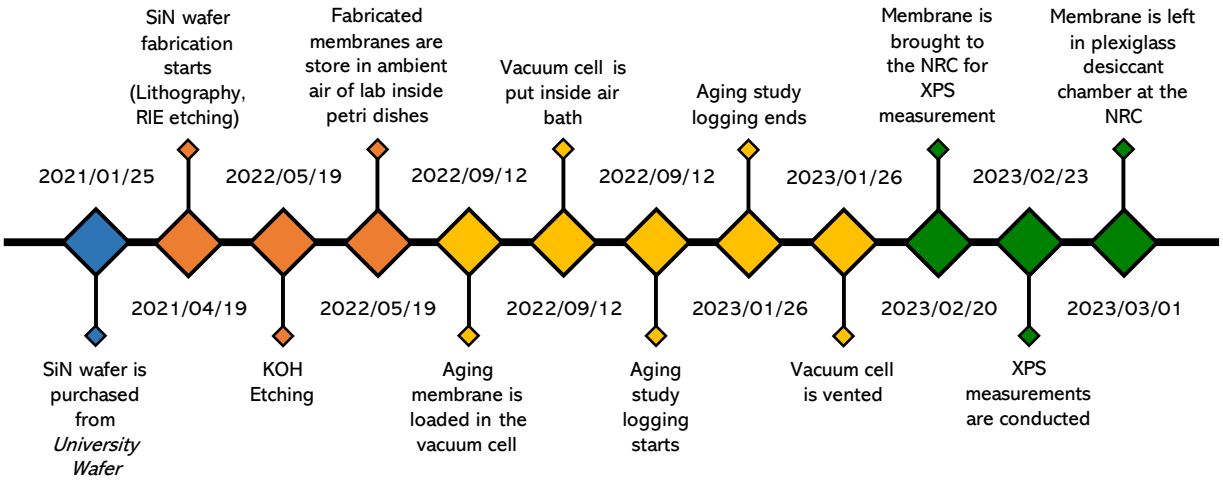


Figure 45: Timeline of the journey for our samples during the research.

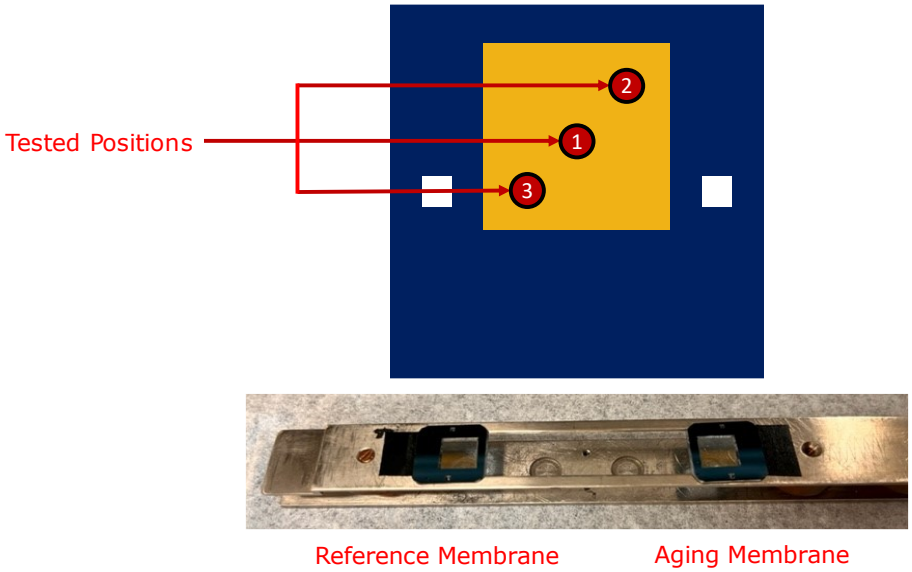


Figure 46: XPS measurements setup and studied positions.

Preliminary results of the XPS measurements showcased that the reference sample, which was never placed under vacuum, had higher concentrations and species of contaminants. As depicted in the preliminary results for the reference membrane in **Figure 47**, the major contaminants found were sodium (Na), calcium (Ca), and potassium (K). Other contaminants, present in lesser concentrations, included chlorine (Cl) as well as fluorine (F). Magnesium (Mg) and copper (Cu) were present in such small magnitudes that they were in the noise error margin of the XPS machine. The large presence of silicon (Si) and nitrogen (N) was expected, since our SiN

membranes are fabricated from these elements. Oxygen (O) and carbon (C) were present in large quantities, but these elements are expected at such high magnitudes in any XPS measurement. As seen on the graph for the aged membrane, which spent a long time in vacuum, the concentration of the major contaminants is greatly reduced and some of the species are absent from the measurements, which would suggest that they have been desorbed by vacuum, or that the reference membrane kept accumulating excess contaminants over the aging measurement period.

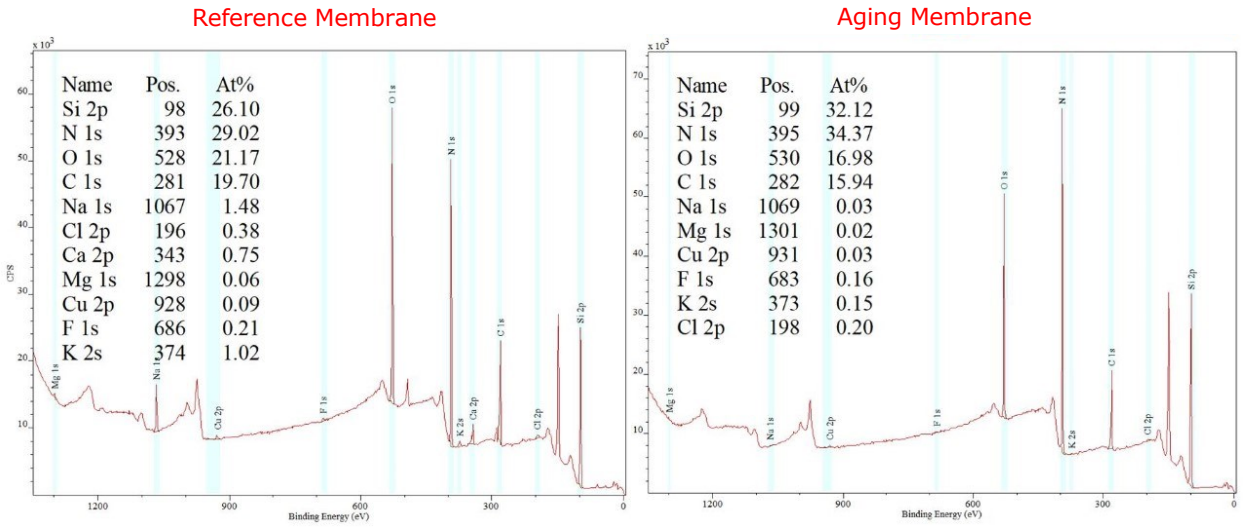


Figure 47: XPS preliminary results comparing reference and aged membrane.

While we know that the aged membrane is much less contaminated than its reference counterpart placed in an ambient setting, wse do not necessarily know the origin of these contaminants. However, we are able to establish educated hypotheses to explain some of their origins:

- Potassium (K): Knowing we use a basic solution of potassium hydroxide (KOH) to etch our membranes, we believe that this contaminant is a remnant of the etch process and could not be removed with our cleaning methods,

- Sodium (Na) and Calcium (Ca): Positive ions of these elements are present in ambient air, under the form of salts. It is possible that trapped negative charges in SiN (which is frequent in dielectric materials) progressively attract these elements over time,
- Fluorine (F): The Petri dishes used to transport the samples are likely the source of fluorine contamination, as suspected by the XPS technician at the NRC who has seen similar results in other situations.

The in-depth analysis of the XPS results for each position are represented for the reference membrane in **Table 3** and for membrane used in the aging study in the **Table 4**. These tables are filled with extensive information reporting the concentration of the elements appearing on the photoelectron spectra, and this, for each analyzed position of both samples. **Table 5** offers a summarized version of the results which gives the average concentration of contaminants for all tested positions on both samples. A colour legend is also implemented to quickly interpret the results: red indicates a higher concentration of contaminants in contrast with green which indicates a lesser concentration, while blue indicates an equal concentration for both samples. It's observed that for the different positions, the concentration of the specie can vary by a large margin. However, by averaging the disparity of results between the positions, we conclude, in essence, that despite the position, the major contaminants are significantly eliminated when the device is placed under vacuum for a large period of time. Another observation is that the presence of carbon increases in the aged sample used in our measurement. It is hypothesized by our collaborator at NRC that repeatedly changing the environment of an object between vacuum and ambient favours the adsorption of carbon during venting, when no water is present on the sample. This phenomenon could explain why our aged membrane has a higher concentration of carbon than the reference sample.

Table 3: Reference membrane XPS results for different tested positions.

Reference Membrane												
		Transition										
Position #	Quantity	Si 2p	N 1s	O 1s	C 1s	Na 1s	Cl 2p	Ca 2p	Cu 2p	K 2s	F 1s	Mg 1s
1	Concentration (%)	26.1	29.0	21.1	19.7	1.5	0.4	0.8	0.1	1.0	0.2	0.1
	Fit Stdev (%)	0.12	0.15	0.14	0.20	0.04	0.05	0.03	0.03	0.11	0.07	0.01
2	Concentration (%)	33.2	39.2	16.8	10.0	0.2	0.3	0.1	0.1	0.0	-	-
	Fit Stdev (%)	0.13	0.15	0.12	0.26	0.02	0.08	0.03	0.03	0.08	-	-
3	Concentration (%)	31.9	37.5	18.6	10.3	0.4	0.5	0.2	0.1	0.4	0.1	0.1
	Fit Stdev (%)	0.12	0.15	0.11	0.21	0.03	0.06	0.03	0.03	0.12	0.07	0.01
Total Positions Average (%)		30.4	35.2	18.9	13.4	0.7	0.4	0.4	0.1	0.5	0.1	0.1
Position Standard Deviation (%)		3.8	5.4	2.2	5.5	0.7	0.1	0.3	0.0	0.5	0.1	0.0
Relative Standard Deviation (%)		12.4	15.4	11.5	41.2	97.6	19.9	97.5	11.2	105.2	57.4	11.9

Table 4: Aged membrane XPS results for different tested positions.

Aging Membrane												
		Transition										
Position #	Quantity	Si 2p	N 1s	O 1s	C 1s	Na 1s	Cl 2p	Ca 2p	Cu 2p	K 2s	F 1s	Mg 1s
1	Concentration (%)	31.7	34.8	17.0	16.0	-	0.4	-	0.0	0.0	0.0	0.0
	Fit Stdev (%)	0.13	0.15	0.12	0.22	-	0.06	-	0.01	0.10	0.05	0.01
2	Concentration (%)	31.8	35.4	16.7	15.8	-	0.2	-	0.0	0.1	-	-
	Fit Stdev (%)	0.13	0.16	0.13	0.21	-	0.07	-	0.02	0.11	-	-
3	Concentration (%)	32.1	34.4	17.0	15.9	-	0.2	-	0.0	0.1	0.2	0.0
	Fit Stdev (%)	0.14	0.17	0.13	0.23	-	0.07	-	0.02	0.10	0.05	0.01
Total Positions Average (%)		31.9	34.9	16.9	15.9	-	0.3	-	0.0	0.1	0.1	0.0
Position Standard Deviation (%)		0.2	0.5	0.1	0.1	-	0.1	-	0.0	0.1	0.1	0.0
Relative Standard Deviation (%)		0.6	1.4	0.8	0.7	-	31.7	-	27.7	62.0	75.9	31.6

Table 5: XPS results summary for average contaminant concentration of both tested samples.

Transition	Si 2p	N 1s	O 1s	C 1s	Na 1s	Cl 2p	Ca 2p	Cu 2p	K 2s	F 1s	Mg 1s
Reference Membrane Total Positions Average (%)	30.4	35.2	18.9	13.4	0.7	0.4	0.4	0.1	0.5	0.1	0.1
Aged Membrane Total Positions Average (%)	31.9	34.9	16.9	15.9	-	0.3	-	0.0	0.1	0.1	0.0

The final XPS measurement that was recorded was the concentration of the contaminants for the position 2 at different X-ray angles of incidence. Having the detector positioned at 90° in relation to the surface of the membrane maximizes the depth of penetration of the projected X-ray waves and thus allows the measurements to gather information for deeper surface layers of our

membranes. By changing the incidence angle, as represented in **Figure 48**, we reduce the depth of penetration of the X-ray wave. This allows us to gather results of contaminants in higher layers of the surface of our membranes. **Table 6** showcases the contaminant concentration results as a function of incidence angles of 0°, 15°, 30°, 45° and 60°. The results obtained do not showcase any obvious trend from which we can extract an existing correlation. Some contaminants seem to have a higher concentration at low angles of incidence indicating that they are more prominent in deeper layers of the surface, while others seem to have a higher concentration at larger angles of incidence, indicating that they are more prominent in upper layers of the surface.

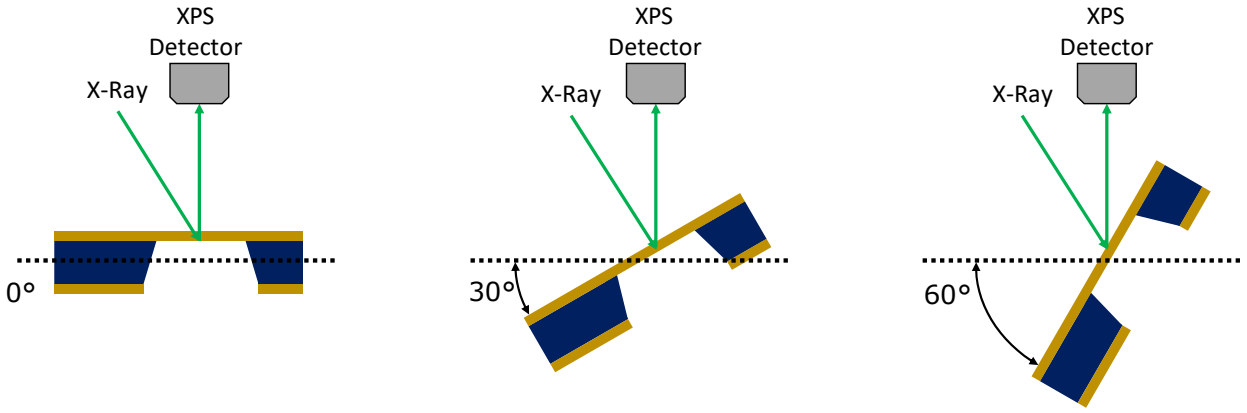


Figure 48: Angle resolved XPS measurement schematic.

Table 6: Angle-resolved XPS results for both tested sample at position #2

Position #2		Reference Membrane									Aging Membrane							Difference			
		Transition									Transition							Transition			
Angle °	Quantity	Si 2p	N 1s	O 1s	C 1s	Na 1s	Cl 2p	Ca 2p	Cu 2p	K 2s	Si 2p	N 1s	O 1s	C 1s	Cl 2p	Cu 2p	K 2s	Si 2p	N 1s	O 1s	C 1s
0	Concentration (%)	33.4	39.3	16.9	9.5	0.2	0.4	0.1	0.1	0.0	31.8	35.4	16.7	15.8	0.2	0.0	0.1	-1.5	-3.9	-0.2	6.2
	Fit Stdev (%)	0.12	0.15	0.13	0.18	0.03	0.08	0.03	0.03	0.09	0.12	0.15	0.14	0.23	0.07	0.02	0.10	0.17	0.22	0.18	0.29
15	Concentration (%)	34.2	38.5	16.3	10.5	0.1	0.2	0.1	0.1	0.0	32.2	34.6	16.4	16.4	0.3	0.0	0.0	-2.0	-3.9	0.1	5.9
	Fit Stdev (%)	0.13	0.15	0.12	0.21	0.03	0.06	0.03	0.03	0.09	0.13	0.15	0.12	0.21	0.06	0.02	0.09	0.18	0.21	0.17	0.30
30	Concentration (%)	34.5	35.9	17.4	11.5	0.1	0.3	0.1	0.0	0.2	31.9	33.4	17.1	17.4	0.3	0.0	-0.1	-2.6	-2.5	-0.3	5.9
	Fit Stdev (%)	0.15	0.16	0.14	0.22	0.02	0.06	0.03	0.03	0.10	0.13	0.16	0.13	0.21	0.07	0.02	0.09	0.20	0.23	0.19	0.30
45	Concentration (%)	34.4	31.9	19.6	13.2	0.2	0.4	0.1	0.1	0.1	31.6	29.3	18.5	20.3	0.2	0.1	0.1	-2.9	-2.7	-1.1	7.1
	Fit Stdev (%)	0.16	0.17	0.17	0.23	0.03	0.09	0.04	0.03	0.10	0.14	0.17	0.14	0.23	0.07	0.02	0.10	0.21	0.24	0.22	0.33
60	Concentration (%)	31.0	27.6	21.5	19.0	0.1	0.2	0.1	0.1	0.3	29.0	23.4	20.4	27.0	0.3	0.0	-0.1	-2.0	-4.2	-1.1	8.0
	Fit Stdev (%)	0.18	0.17	0.16	0.26	0.03	0.07	0.04	0.03	0.11	0.17	0.18	0.16	0.29	0.08	0.02	0.12	0.25	0.25	0.23	0.39

5. Work Conclusion

5.1 Findings Recapitulation

This work is a first measurement of long-term stability of silicon nitride (SiN) MEMS devices, a crucial information that can be used to compensate our measurements and calibrate devices used over long periods of time as sensors. The observed aging in this work, for our 6 × 6 mm SiN membrane, is of 300 ppm for 135 days. The aging trend observed for our resonator was very similar to the double logarithmic models found in literature, for other types of oscillators. A hybrid logarithmic and linear model was also able to represent the aging of our device. The initial aging drift observed follows the aging model proposed in this work, which stipulates that the adsorption of water contaminants on our membrane is one of the main causes of aging. The beginning of the studied aging follows a trend of decreasing frequency. This behaviour is coherent with the desorption of the contaminating water molecules, due to the vacuum environment, which were adsorbed when the membrane was stored in ambient setting prior to the beginning of our studies. The reviewed aging model shows that contamination by water adsorption had its own sub-mechanisms that each lead to aging of our membrane:

- Slight aging was induced by a mass increase resulting from adsorption of water molecules on our membrane,
- Moderate aging was induced by a stress source issued from the water surface tension of the adsorbed contaminant on our membrane,
- The most important aging was induced by a stress source issued from the chemical reaction between the adsorbed water and our membrane.

While we have not obtained a stability of 1 ppm/year, a metric desired for the National Research Council of Canada (NRC), we have reported the first long-term stability trend for a SiN resonating device, which is pivotal for future development of applications using this platform. The drift amount due to aging of our membrane might seem much more important than the reviewed quartz and silicon oscillators, but this is due to the extremely small thickness of our membrane. When considering the drift and weighing it by the thickness of the device, the aging is around the same order of magnitude as that of quartz and silicon MEMS oscillators. This tends to confirm the commonly accepted assumption that aging of micro/nano resonator is essentially a surface (rather than bulk) phenomenon.

Additionally, X-ray photoelectron spectroscopy (XPS) measurements were conducted by the NRC on our membrane used in our aging experiment as well as a reference membrane that was created at the same time but stored in the ambient environment of our lab. The measurements revealed that the reference membrane contained a few major contaminants such as sodium (Na), potassium (K) and calcium (Ca) as well as minor contaminants like chlorine (Cl) and fluorine (F). The results showcase that the aged membrane, which spent a large amount of time under vacuum, contains a much lesser concentration of these major contaminants, thus suggesting that they have been desorbed in the vacuum environment, and that the reference membrane kept accumulating more of these contaminants during the aging measurement period.

5.2 Future Aging Measurements Improvements and Recommendations

During this work, we have continuously improved our measurement methodology and the physical setup used. While the experiment was very robust, improvements can always be made. The principal change to be made would be to extend the aging measurement period beyond 1 year.

Doing so would have provided a better and more accurate aging trend. Unfortunately, due to the availability of the equipment we use, we could not have gone further than 135 days in our case.

Another change to be done would be to omit waiting for the piezoelectric actuator's self generated heat to reach steady state and start the data streaming as soon as the phase-locked loop (PPL) could continuously track the membrane's fundamental resonance frequency. The data measured could be compensated for the heating of the piezoelectric actuator and we could gain important insight on the aging the device sustains in the critical initial stages where vacuum is initiated and important drift occurs.

While our devices are cleaned right after they are manufactured, the period they spend in ambient setting before they are used in a vacuum setting, allows many contaminants to be adsorbed on the membrane. Thus, cleaning the membrane once again before setting it in vacuum, even if there are no contaminants visible to the human eye, would be recommended to reduce initial aging of contaminants desorption. Likewise, baking the sample prior to installing it in the chamber could be envisioned to reduce initial contamination further, and storing in an inert gas cabinet should be envisioned.

Finally, it would be important to gain knowledge on other factors that could impact the aging of our SiN devices. To do so, setting up these aging experiments for 1×1 mm, 1.5×1.5 mm and 3×3 mm membranes, would allow us to gather insightful information on how the geometry of our devices affects their aging. Furthermore, fabricating membranes with stochiometric high-stress Si_3N_4 (1 GPa of internal stress) rather than in silicon rich low-stress SiN_x (0.1 GPa) membranes used in our work and studying their aging could lead to better performances or reveal key information on how the structure and material composition of our MEMS affects its aging.

References

[1] C. Wang and J. Park, "Magnetic micropump embedded in contact lens for on-demand drug delivery," *Micro and Nano Syst Lett*, vol. 8, no. 1, p. 1, Dec. 2020, doi: 10.1186/s40486-019-0101-x.

[2] C. Barthold, K. Pathapati Subbu, and R. Dantu, "Evaluation of gyroscope-embedded mobile phones," in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, Oct. 2011, pp. 1632–1638. doi: 10.1109/ICSMC.2011.6083905.

[3] R. P. Middlemiss, A. Samarelli, D. J. Paul, J. Hough, S. Rowan, and G. D. Hammond, "Measurement of the Earth tides with a MEMS gravimeter," *Nature*, vol. 531, no. 7596, pp. 614–617, Mar. 2016, doi: 10.1038/nature17397.

[4] M. Lutz *et al.*, "MEMS Oscillators for High Volume Commercial Applications," in *TRANSDUCERS 2007 - 2007 International Solid-State Sensors, Actuators and Microsystems Conference*, Jun. 2007, pp. 49–52. doi: 10.1109/SENSOR.2007.4300068.

[5] G. Zhanshe, "Research development of silicon MEMS gyroscopes: a review," *Microsyst Technol*, p. 14, 2015.

[6] Z. Chang, "Radiative Heat Transfer in Freestanding Silicon Nitride Membranes," p. 7, 2020.

[7] X. C. Zhang, E. B. Myers, J. E. Sader, and M. L. Roukes, "Nanomechanical Torsional Resonators for Frequency-Shift Infrared Thermal Sensing," *Nano Lett.*, vol. 13, no. 4, pp. 1528–1534, Apr. 2013, doi: 10.1021/nl304687p.

[8] J. R. Vig and Y. Kim, "Noise in microelectromechanical system resonators," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 46, no. 6, pp. 1558–1565, Nov. 1999, doi: 10.1109/58.808881.

[9] A. N. Cleland and M. L. Roukes, "Noise processes in nanomechanical resonators," *Journal of Applied Physics*, vol. 92, no. 5, pp. 2758–2769, Sep. 2002, doi: 10.1063/1.1499745.

[10] P. Sadeghi, A. Demir, L. G. Villanueva, H. Kähler, and S. Schmid, "Frequency fluctuations in nanomechanical silicon nitride string resonators," *Phys. Rev. B*, vol. 102, no. 21, p. 214106, Dec. 2020, doi: 10.1103/PhysRevB.102.214106.

[11] C. Zhang and R. St-Gelais, "Demonstration of Frequency Stability limited by Thermal Fluctuation Noise in Silicon Nitride Nanomechanical Resonators." arXiv, Jan. 20, 2023. Accessed: Mar. 22, 2023. [Online]. Available: http://arxiv.org/abs/2211.12651

[12] S. Schmid, L. G. Villanueva, and M. L. Roukes, *Fundamentals of Nanomechanical Resonators*. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-28691-4.

[13] "RECOMMENDATION ITU-R TF.686-3* - Glossary and definitions of time and frequency terms," p. 20.

[14] T. Hodges *et al.*, "Characterization of Mass-Loaded Silicon Nitride On-Chip Resonators for Traceable Sensing of Low Amplitude Acceleration," in *2022 IEEE International Symposium on Inertial Sensors and Systems (INERTIAL)*, May 2022, pp. 1–4. doi: 10.1109/INERTIAL53425.2022.9787524.

[15] D. Chen, Z. Wu, X. Shi, J. Wang, and L. Liu, "Design and modeling of an electromagnetically excited silicon nitride beam resonant pressure sensor," in *2009 4th IEEE International Conference on Nano/Micro Engineered and Molecular Systems*, Jan. 2009, pp. 754–757. doi: 10.1109/NEMS.2009.5068688.

[16] T. Larsen *et al.*, “Ultrasensitive string-based temperature sensors,” *Appl. Phys. Lett.*, vol. 98, no. 12, p. 121901, Mar. 2011, doi: 10.1063/1.3567012.
[17] R. Kubo, “The fluctuation-dissipation theorem,” p. 31.
[18] B. M. Zwickl *et al.*, “High quality mechanical and optical properties of commercial silicon nitride membranes,” *Appl. Phys. Lett.*, vol. 92, no. 10, p. 103125, Mar. 2008, doi: 10.1063/1.2884191.
[19] M. Bao, *Analysis and Design Principles of MEMS Devices*. Elsevier, 2005.
[20] D. Ruffieux, F. Krummenacher, A. Pezous, and G. Spinola-Durante, “Silicon Resonator Based 3.2 µW Real Time Clock With $\pm$10 ppm Frequency Accuracy,” *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 224–234, Jan. 2010, doi: 10.1109/JSSC.2009.2034434.
[21] J. R. Vig and T. R. Meeker, “The aging of bulk acoustic wave resonators, filters and oscillators,” in *Proceedings of the 45th Annual Symposium on Frequency Control 1991*, May 1991, pp. 77–101. doi: 10.1109/FREQ.1991.145888.
[22] H. Zhou, C. Nicholls, T. Kunz, and H. Schwartz, “Frequency Accuracy & Stability Dependencies of Crystal Oscillators,” 2008.
[23] “Quartz Crystal Cuts: AT, BT, SC, CT . . » Electronics Notes.” https://www.electronics-notes.com/articles/electronic_components/quartz-crystal-xtal/crystal-resonator-cuts-at-bt-sc-ct.php (accessed Jun. 20, 2023).
[24] “Quartz crystal device.” https://www.qiaj.jp/pages/frame20/page01-e.html (accessed Apr. 11, 2023).
[25] P. T. Landsberg, “On the Logarithmic Rate Law in Chemisorption and Oxidation,” *The Journal of Chemical Physics*, vol. 23, no. 6, pp. 1079–1087, Jun. 1955, doi: 10.1063/1.1742193.
[26] R. L. Filler and J. R. Vig, “Long-term aging of oscillators,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 40, no. 4, pp. 387–394, Jul. 1993, doi: 10.1109/58.251287.
[27] D. E. Pierce, R. A. Murray, R. Lareau, S. Laffey, and J. R. Vig, “Outgassing of quartz,” in *Proceedings of IEEE 48th Annual Symposium on Frequency Control*, Jun. 1994, pp. 107–114. doi: 10.1109/FREQ.1994.398348.
[28] S. Łuczak, J. Wierciak, and W. Credo, “Effects of Natural Aging in Biaxial MEMS Accelerometers,” *IEEE Sensors Journal*, vol. 21, no. 2, pp. 1305–1314, Jan. 2021, doi: 10.1109/JSEN.2020.3017897.
[29] F. L. Walls and J. R. Vig, “Fundamental limits on the frequency stabilities of crystal oscillators,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 42, no. 4, pp. 576–589, Jul. 1995, doi: 10.1109/58.393101.
[30] “Technology Paper: SiTime’s MEMS First™ and EpiSeal™ Processes | SiTIme.” https://www.sitime.com/support/resource-library/technology-papers/sitimes-mems-firsttm-and-episealtm-processes (accessed Feb. 06, 2023).
[31] Y. Artioli, “Adsorption,” in *Encyclopedia of Ecology*, S. E. Jørgensen and B. D. Fath, Eds., Oxford: Academic Press, 2008, pp. 60–65. doi: 10.1016/B978-008045405-4.00252-4.
[32] A. C. Stephan, E. L. Finot, H.-F. Ji, L. A. Pinnaduwage, and T. Thundat, “Micromechanical measurement of active sites on silicon nitride using surface free energy variation,” *Ultramicroscopy*, vol. 91, no. 1–4, pp. 1–8, May 2002, doi: 10.1016/S0304-3991(02)00076-1.

[33] "Moist Air - Relative Humidity." https://www.engineeringtoolbox.com/relative-humidity-air-d_687.html (accessed Mar. 06, 2023).
[34] "The effects of magnetic fields on water molecular hydrogen bonds," *Journal of Molecular Structure*, vol. 938, no. 1–3, pp. 15–19, Dec. 2009, doi: 10.1016/j.molstruc.2009.08.037.
[35] N. Nadermann, C.-Y. Hui, and A. Jagota, "Solid surface tension measured by a liquid drop under a solid film," *Proceedings of the National Academy of Sciences*, vol. 110, no. 26, pp. 10541–10545, Jun. 2013, doi: 10.1073/pnas.1304587110.
[36] G. Mu *et al.*, "Observation of Silicon Nitride Nanomechanical Resonator Actuation Using Capacitive Substrate Excitation." arXiv, Dec. 16, 2021. doi: 10.48550/arXiv.2112.09303.
[37] "Custom IC Layout," *Siemens Digital Industries Software*. https://eda.sw.siemens.com/en-US/ic/ic-custom/ams/l-edit-ic/ (accessed Jan. 12, 2023).
[38] "Crystalbond™ Adhesives." https://www.tedpella.com/technote_html/821-1-2-3-4-6-TN.pdf (accessed Feb. 13, 2023).
[39] "Vacuum Baking Process – What is it and why would you choose it for your application?," *UC Components*, Dec. 10, 2019. https://www.uccomponents.com/vacuum-baking-process-what-is-it-and-why-would-you-choose-it-for-your-application/ (accessed Mar. 15, 2023).
[40] M. Giroux, C. Zhang, N. Snell, G. Mu, M. Stephan, and R. St-Gelais, "High resolution measurement of near-field radiative heat transfer enabled by nanomechanical resonators," *Appl. Phys. Lett.*, vol. 119, no. 17, p. 173104, Oct. 2021, doi: 10.1063/5.0068700.
[41] "16059,16059-10 TN.pdf." Accessed: Feb. 13, 2023. [Online]. Available: https://www.tedpella.com/technote_html/16059,16059-10%20TN.pdf
[42] R. Nikbakht, X. Xie, A. Weck, and R. St-Gelais, "High quality factor silicon nitride nanomechanical resonators fabricated by maskless femtosecond laser micromachining," *Journal of Vacuum Science & Technology B*, vol. 41, no. 2, p. 023002, Mar. 2023, doi: 10.1116/5.0124150.
[43] D. Rugar, H. J. Mamin, and P. Guethner, "Improved fiber-optic interferometer for atomic force microscopy," *Appl. Phys. Lett.*, vol. 55, no. 25, pp. 2588–2590, Dec. 1989, doi: 10.1063/1.101987.
[44] "Phase-Locked Loops for Analog Signals." https://www.zhinst.com/sites/default/files/documents/2022-09/zi_whitepaper_phase_locked_loop.pdf (accessed Feb. 28, 2023).
[45] J. Janesch, "Two-Wire vs. Four-Wire Resistance Measurements: Which Configuration Makes Sense for Your Application?", Accessed: Feb. 28, 2023. [Online]. Available: https://assets.testequity.com/te1/Documents/ARTICLE_LIBRARY/Keithley%20-Two-Wire%20vs%20Four-Wire%20Resistance%20Measurements.pdf

Appendices

A. Python Scripts

Continuous Recording.py

```
#============================================ A - Importing Libraries
================================================#
print("\nRead 'Continuous Recording README.txt' for any information about
this script.")
print('\nLoading Libraries...')
import time                                             #Import library for
timing and timestamps
import numpy as np                                      #Import library for
array and basic mathematical manipulations
import pandas as pd                                     #Import library for
large array manipulations
import pyvisa as visa                                   #Import library to
control Multimeter
import zhinst.utils as Zu                               #Import library to
control Lock-In Amplifier
from datetime import datetime                           #Import library for
timing, timestamps and dating
print('\nLibraries Loaded!\n')
#=========================================================================
=======================================#

#============================================ B - Data Stream Function
================================================#
def
read_data_update_plot(timestamp0,dct_Storage_LI,dct_Storage_M,dmm,Initial_Tim
e):
    data_read = data_acquisition.read(True)
    returned_signal_paths = [signal_path.lower() for signal_path in
data_read.keys()]
    progress = data_acquisition.progress()[0]
    # Loop over all the subscribed signals:
    for signal_path in signal_paths:
        dct_Storage_M[0] = np.append(dct_Storage_M[0],
float(dmm.query(Multimeter_Measure)))
        dct_Storage_M[1] = np.append(dct_Storage_M[1], (datetime.now()-
Initial_Time).total_seconds())
        if signal_path.lower() in returned_signal_paths:
            # Loop over all the bursts for the subscribed signal. More than
            # one burst may be returned at a time, in particular if we call
            # read() less frequently than the burst_duration.
            for index, signal_burst in
enumerate(data_read[signal_path.lower()]):
                if np.any(np.isnan(timestamp0)):
                    # Set our first timestamp to the first timestamp we
obtain.
                    timestamp0 = signal_burst["timestamp"][0, 0]
```

```
                # Convert from device ticks to time in seconds.
                t = (signal_burst["timestamp"][0, :]-timestamp0)/clockbase
                value = signal_burst["value"][0, :]

                if signal_path == signal_paths[0]:
                    dct_Storage_LI[0] =
np.append(dct_Storage_LI[0],np.array(value))
                    dct_Storage_LI[1] =
np.append(dct_Storage_LI[1],np.array(t))
                elif signal_path == signal_paths[1]:
                    dct_Storage_LI[2] =
np.append(dct_Storage_LI[2],np.array(value))
                elif signal_path == signal_paths[2]:

dct_Storage_LI[3]=np.append(dct_Storage_LI[3],np.array(value))
                    dct_Storage_LI[4] =
np.append(dct_Storage_LI[4],np.array(t))
                elif signal_path == signal_paths[3]:

dct_Storage_LI[5]=np.append(dct_Storage_LI[5],np.array(value))
                elif signal_path == signal_paths[4]:

dct_Storage_LI[6]=np.append(dct_Storage_LI[6],np.array(value))
                    dct_Storage_LI[7] =
np.append(dct_Storage_LI[7],np.array(t))

        else:
            # Note: If we read before the next burst has finished, there may
be no new data.
            # No action required.
            pass

    return timestamp0,dct_Storage_LI,dct_Storage_M
#================================================================================
=========================================#

#============================================== 1 - Device Connection
================================================#
print(visa.ResourceManager().list_resources())
resource_manager = visa.ResourceManager()
inst_resource_string_1 = "TCPIP0::169.254.209.4::inst0::INSTR"          #LAN
#inst_resource_string_1 = "USB0::0x05E6::0x6500::04484485::INSTR"       #USB
dmm = resource_manager.open_resource(inst_resource_string_1)
dmm.clear()

ID=dmm.query("*IDN?")
print('The IDN of the multimeter is:',ID)
device_id = 'dev5855'
apilevel_example = 6
(daq, device, _) = Zu.create_api_session(device_id, apilevel_example)
#================================================================================
=========================================#

#======================================== 2 - Initiating Lock-In Signals
==============================================#
L_Dat_1_Path = f"/{device}/demods/0/sample.frequency"
L_Dat_2_Path = f"/{device}/demods/0/sample.r"
```

```
L_Dat_3_Path = f"/{device}/demods/1/sample.frequency"
L_Dat_4_Path = f"/{device}/demods/1/sample.r"
L_Dat_5_Path = f"/{device}/demods/2/sample.r"

signal_paths = []
signal_paths.append(L_Dat_1_Path)
signal_paths.append(L_Dat_2_Path)
signal_paths.append(L_Dat_3_Path)
signal_paths.append(L_Dat_4_Path)
signal_paths.append(L_Dat_5_Path)
#=========================================================================
=======================================#

#================================= 3 - Initiating Multimeter Signal and
Setting ======================================#
dmm.write("SENSe:FRESistance:RANGe 1e5, (@2)")
dmm.write("SENSe:RESistance:RANGe 1e4, (@3)")

Multimeter_Measure = ':MEASure:FRESistance?'
Multimeter_Measure_2 = ':MEASure:RESistance?'
#=========================================================================
=======================================#

#================================== 4 - Setting Up Data Storage Structure
=======================================#
F_Mode_11 = []
T_Mode_11 = []
V_Mode_11 = []
F_Mode_21 = []
T_Mode_21 = []
V_Mode_21 = []
V_Fringe = []
T_Fringe = []
dct_Storage_LI={0:F_Mode_11,1:T_Mode_11,2:V_Mode_11,3:F_Mode_21,4:T_Mode_21,5
:V_Mode_21,6:V_Fringe,7:T_Fringe}

#Initializing Multimeter Thermistor Arrays
M_Dat_1 = []
#Resistance of Thermistor in Vacuum Cell
M_Dat_1_T = []
#Time of measurement
dct_Storage_M={0:M_Dat_1,1:M_Dat_1_T}

#Initializing Multimeter Thermistor Arrays
M_Dat_RTD = []
#Resistance of RTD
M_Dat_RTD_T = []
#Time of measurement
dct_Storage_M_RTD={0:M_Dat_1,1:M_Dat_RTD_T}
#=========================================================================
=======================================#

#================================== 5 - Timing & Interval Selector
=============================================#
Current_Time = datetime.now()

#Recording Rate Settings
```

```
total_duration = 60*60*24*31*12
#Total time for data logging in seconds
module_sampling_rate = 7
#Streaming rate (data point per second)
burst_duration = 1
#Lengths of bursts in seconds
num_cols = int(np.ceil(module_sampling_rate * burst_duration))
num_bursts = int(np.ceil(total_duration / burst_duration))

#Recording File Time Limit
Recording_Time_Limit = 60*60*6
Recording_Time_Limit_RTD = 5*60

#Recording Update
Checkpoint_Interval = 60*60*6
Checkpoint_Time = 0
#================================================================================
========================================#

#==================================== 6 - Saving Settings, Paths & Headers
========================================#
Save_Path=r"C:\Users\St-Gelais\OneDrive - University of Ottawa\01 UOMEMS
Student Project Folder - SHARED\Michel Stephan\12 - Stability
Measurement\Cell - 6 mm - Low Stress\Lock-In With Multimeter"
LockIn_Save_Path=Save_Path+"\LockIn_"
S=" ; "
dct_h_LI={0:'DM1 Frequency [Hz]',1:'DM1 F Time [s]',2:'DM1 Voltage
[V]',3:'DM2 Frequency [Hz]',4:'DM2 Time [s]',5:'DM2 Voltage Time [V]',6:'DM3
Voltage [V]',7:'DM3 V Time [s]'}
Partial_Header_LockIn=dct_h_LI[0]+S+dct_h_LI[1]+S+dct_h_LI[2]+S+dct_h_LI[3]+S
+dct_h_LI[4]+S+dct_h_LI[5]+S+dct_h_LI[6]+S+dct_h_LI[7]

dct_h_M={0:'Thermistor 4-W Resistance [Ohm]',1:'Resistance Time  [s]'}
Multimeter_Save_Path=Save_Path+"\Multimeter_"
Partial_Header_Multimeter=dct_h_M[0]+S+dct_h_M[1]

dct_h_M_RTD={0:'RTD 2-W Resistance [Ohm]',1:'Resistance Time  [s]'}
Multimeter_RTD_Save_Path=Save_Path+"\Multimeter_RTD_"
Partial_Header_Multimeter_RTD=dct_h_M_RTD[0]+S+dct_h_M_RTD[1]

File_Index=0
#================================================================================
========================================#

#==================================== 7 - Lock-In Amplifier Streaming Settings
========================================#
data_acquisition = daq.dataAcquisitionModule()
data_acquisition.set("device", device)
data_acquisition.set("type", 0)
data_acquisition.set("grid/mode", 2)
data_acquisition.set("count", num_bursts)
data_acquisition.set("duration", burst_duration)
data_acquisition.set("grid/cols", num_cols)
#================================================================================
========================================#

#==================================== 8 - Lock-In Amplifier Signal Subscribing
```

```
========================================#
print()

# A dictionary to store all the acquired data.
for signal_path in signal_paths:
    print("Subscribing to", signal_path)
    data_acquisition.subscribe(signal_path)

print()

clockbase = float(daq.getInt(f"/{device}/clockbase"))

ts0 = np.nan
read_count = 0

i = 0
#=========================================================================
========================================#

#================================== 9 - RTD Streaming & Overall Data Saving
========================================#
# Start recording data.
data_acquisition.execute()

# Record data in a loop with timeout.
timeout = 1.5 * total_duration
t0_measurement = datetime.now()
# The maximum time to wait before reading out new data.
t_update = 0.09 * burst_duration

Initial_Time = datetime.now()
Recording_Time = datetime.now()
Recording_Time_RTD = datetime.now()

dmm.write(":ROUT:OPEN (@2)")
dmm.write(":ROUT:CLOS (@3)")
dct_Storage_M_RTD[0] = np.append(dct_Storage_M_RTD[0],
float(dmm.query(Multimeter_Measure_2)))
dct_Storage_M_RTD[1] = np.append(dct_Storage_M_RTD[1], (datetime.now() -
Initial_Time).total_seconds())
Recording_Time_RTD = datetime.now()
dmm.write(":ROUT:OPEN (@3)")
dmm.write(":ROUT:CLOS (@2)")

while not data_acquisition.finished():
    t0_loop = datetime.now()
    if (datetime.now() - t0_measurement).total_seconds() > timeout:
        raise Exception(f"Timeout after {timeout} s - recording failed. "
            f"Ensured signals are enabled and properly referred to in the
script")

    ts0, dct_Storage_LI,dct_Storage_M = read_data_update_plot(ts0,
dct_Storage_LI,dct_Storage_M,dmm,Initial_Time)
    read_count += 1

    if (datetime.now()-Recording_Time_RTD).total_seconds() >=
Recording_Time_Limit_RTD:
```

```
        dmm.write(":ROUT:OPEN (@2)")
        dmm.write(":ROUT:CLOS (@3)")
        dct_Storage_M_RTD[0] = np.append(dct_Storage_M_RTD[0],
float(dmm.query(Multimeter_Measure_2)))
        dct_Storage_M_RTD[1] = np.append(dct_Storage_M_RTD[1],
(datetime.now() - Initial_Time).total_seconds())
        Recording_Time_RTD = datetime.now()
        dmm.write(":ROUT:OPEN (@3)")
        dmm.write(":ROUT:CLOS (@2)")

    if (datetime.now()-Recording_Time).total_seconds() >=
Recording_Time_Limit:
        Current_Time=datetime.now()
        Recorded_Data=pd.DataFrame.from_dict(dct_Storage_LI)
        Recorded_Data2=pd.DataFrame.from_dict(dct_Storage_M)
        Recorded_Data3=pd.DataFrame.from_dict(dct_Storage_M_RTD)
        np.savetxt(LockIn_Save_Path + str(File_Index) + '.txt',
Recorded_Data,
delimiter=';',header=Partial_Header_LockIn+S+str(Current_Time), comments='',
fmt='%s')
        np.savetxt(Multimeter_Save_Path + str(File_Index) + '.txt',
Recorded_Data2,
delimiter=';',header=Partial_Header_Multimeter+S+str(Current_Time),
comments='', fmt='%s')
        np.savetxt(Multimeter_RTD_Save_Path + str(File_Index) + '.txt',
Recorded_Data3,
delimiter=';',header=Partial_Header_Multimeter_RTD+S+str(Current_Time),
comments='', fmt='%s')
        Recording_Time = datetime.now()

        #Initializing Lock-In Amplifier Arrays
        F_Mode_11 = []
        T_Mode_11 = []
        V_Mode_11 = []
        F_Mode_21 = []
        T_Mode_21 = []
        V_Mode_21 = []
        V_Fringe = []
        T_Fringe = []

dct_Storage_LI={0:F_Mode_11,1:T_Mode_11,2:V_Mode_11,3:F_Mode_21,4:T_Mode_21,5
:V_Mode_21,6:V_Fringe,7:T_Fringe}

        #Initializing Multimeter Arrays
        M_Dat_1 = []
        M_Dat_1_T = []
        M_Dat_2 = []
        dct_Storage_M={0:M_Dat_1,1:M_Dat_1_T}

        # Initializing Multimeter Thermistor Arrays
        M_Dat_RTD = []  # Resistance of RTD
        M_Dat_RTD_T = []  # Time of measurement
        dct_Storage_M_RTD = {0: M_Dat_1, 1: M_Dat_RTD_T}

        print("Data saved in file with index %d" %(File_Index))
        File_Index = File_Index + 1
```

```
    if (datetime.now()-Initial_Time).total_seconds() > Checkpoint_Time:
        print("Currently Recording. %s" %(datetime.now()))
        Checkpoint_Time = Checkpoint_Time + Checkpoint_Interval

        if Zu.api_server_version_check(daq) != True:
            print("Connection error. Data logging stopped")
            print("Breaking Loop")
            break

    time.sleep(max(0, t_update - (datetime.now() - t0_loop).total_seconds()))
#=============================================================================
=========================================#

#=================================== 10 - End Data Streaming & Save Final
Data ======================================#
_, dct_Storage_LI,dct_Storage_M = read_data_update_plot(ts0,
dct_Storage_LI,dct_Storage_M,dmm,Initial_Time)


dmm.write(":ROUT:OPEN (@2)")
dmm.write(":ROUT:CLOS (@3)")
dct_Storage_M_RTD[0] = np.append(dct_Storage_M_RTD[0],
float(dmm.query(Multimeter_Measure_2)))
dct_Storage_M_RTD[1] = np.append(dct_Storage_M_RTD[1], (datetime.now() -
Initial_Time).total_seconds())
Recording_Time_RTD = datetime.now()
dmm.write(":ROUT:OPEN (@3)")

Recorded_Data=pd.DataFrame.from_dict(dct_Storage_LI)
Recorded_Data2=pd.DataFrame.from_dict(dct_Storage_M)
Recorded_Data3=pd.DataFrame.from_dict(dct_Storage_M_RTD)
np.savetxt(LockIn_Save_Path+ str(File_Index) +
'.txt',Recorded_Data,delimiter=';',header=Partial_Header_LockIn+S+str(Current
_Time),comments='', fmt='%s')
np.savetxt(Multimeter_Save_Path + str(File_Index) +
'.txt',Recorded_Data2,delimiter=';',header=Partial_Header_Multimeter+S+str(Cu
rrent_Time),comments='', fmt='%s')
np.savetxt(Multimeter_RTD_Save_Path+str(File_Index) + '.txt', Recorded_Data3,
delimiter=';',header=Partial_Header_Multimeter_RTD+S+str(Current_Time),
comments='', fmt='%s')

print("Final data saved in file with index %d" %(File_Index))

dmm.close()
#=============================================================================
=========================================#
```

Merging and Filtering.py

```
#============================================== A - Importing Libraries
================================================#
print("\nRead 'Merging and Filtering README.txt' for any information about
this script.")
print('\nLoading Libraries...')
import glob
import datefinder
import numpy as np
import pandas as pd
from tkinter import *
import scipy.signal as sign
from functools import partial
import matplotlib.pyplot as plt
from colorama import Fore, Back, Style
from datetime import datetime, timedelta
print('\nLibraries Loaded!\n')
#============================================================================
=========================================#

#========================================== B - Time Formating Function
================================================#
def Get_Time_Formating(Timestamp):
    if isinstance(Timestamp,float) == True:
        if int(Timestamp) > 259200:Timestamp_2 = 'days'
        elif int(Timestamp) > 18000: Timestamp_2 = 'hours'
        elif int(Timestamp) > 3600: Timestamp_2 = 'minutes'
        else: Timestamp_2 = 'seconds'

        Timestamp=Timestamp_2

    if Timestamp == 'minutes':
        Timescale=60
        TimeLabeling='Time [min]'
    elif Timestamp == 'hours':
        Timescale=60*60
        TimeLabeling='Time [hr]'
    elif Timestamp == 'days':
        Timescale=60*60*24
        TimeLabeling='Time [days]'
    else:
        Timescale=1
        TimeLabeling='Time [s]'

    return Timescale, TimeLabeling
#============================================================================
=========================================#

#========================================== C - GUI Plot Updater Function
================================================#
def Plot_GUI_Update(dicts):
    global Boxes_Status
    global Timestamp

    Boxes_Status=[]
```

```
    Timestamp=Timestamp_Value.get()

    for k in range(len(keys)):
        Boxes_Status.append(dicts[k].get())
#=============================================================================
=========================================#

#======================================== D - GUI Save Settings Function
=============================================#
def Save():
    global Rolling_Median_Factor
    global Trim_Start_Absolute
    global Trim_End_Absolute

    Timescale, TimeLabeling = Get_Time_Formating(Timestamp)

    Rolling_Median_Factor = Rolling_Median.get()
    Trim_Start_Absolute = Time_Begin.get()*Timescale
    Trim_End_Absolute = Time_End.get()*Timescale
#=============================================================================
=========================================#

#============================================== E - GUI Exti Function
=================================================#
def Exit():
    window.destroy()
#=============================================================================
=========================================#

#================================== F - GUI Quick Plotter Interface Function
=========================================#
def Plotting(df_LI,df_MM,df_MM_RTD,Number_Of_Columns,Labels,Import_Lab_RTD):
    df_LI_Filtering = pd.DataFrame()
    df_LI_Prime = pd.DataFrame()
    df_LI_Filtering = df_LI
    Timescale,TimeLabeling = Get_Time_Formating(Timestamp)

    global Trim_Start
    global Trim_End
    global Temporary_Rolling_Median_Factor

    Temporary_Rolling_Median_Factor = Rolling_Median.get()

    Trim_Start=Time_Begin.get()
    Trim_End=Time_End.get()

    if Trim_Start != 0:
        Trim_Start_Absolute = Trim_Start * Timescale
        df_LI = df_LI.loc[df_LI[1] > Trim_Start_Absolute]
        df_MM = df_MM.loc[df_MM[1] > Trim_Start_Absolute]

        if Import_Lab_RTD == 'y':
            df_MM_RTD = df_MM_RTD.loc[df_MM_RTD[1] > Trim_Start_Absolute]

    if Trim_End > Trim_Start:
        Trim_End_Absolute = Trim_End * Timescale
        df_LI = df_LI.loc[df_LI[1] < Trim_End_Absolute]
```

```
        df_MM = df_MM.loc[df_MM[1] < Trim_End_Absolute]

        if Import_Lab_RTD == 'y':
            df_MM_RTD = df_MM_RTD.loc[df_MM_RTD[1] < Trim_End_Absolute]

    if Temporary_Rolling_Median_Factor != 0:
        print(Print_Format+"Testing Rolling Median with factor:
%d"%(Temporary_Rolling_Median_Factor))

    limit = 0.01
    df_LI_Filtering[Number_Of_Columns] =
df_LI_Filtering[0].rolling(Temporary_Rolling_Median_Factor).median()
    df_LI_Filtering[Number_Of_Columns + 1] = df_LI_Filtering[0] -
df_LI_Filtering[Number_Of_Columns]
    df_LI_Filtering[Number_Of_Columns + 1] =
df_LI_Filtering[Number_Of_Columns + 1].abs()
    df_LI_Filtering =
df_LI_Filtering.drop(df_LI_Filtering[df_LI_Filtering[Number_Of_Columns + 1] >
limit].index)

    if Number_Of_Columns == 4:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1].to_numpy()
        df_LI_Prime[2] = np.interp(df_LI[1], df_LI_Filtering[3],
df_LI_Filtering[2])

    if Number_Of_Columns == 8:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1].to_numpy()
        df_LI_Prime[2] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[2])
        df_LI_Prime[3] = np.interp(df_LI[1], df_LI_Filtering[4],
df_LI_Filtering[3])
        df_LI_Prime[4] = np.interp(df_LI[1], df_LI_Filtering[4],
df_LI_Filtering[5])
        df_LI_Prime[5] = np.interp(df_LI[1], df_LI_Filtering[7],
df_LI_Filtering[6])

    if Number_Of_Columns == 10:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1].to_numpy()
        df_LI_Prime[2] = np.interp(df_LI[1], df_LI_Filtering[3],
df_LI_Filtering[2])
        df_LI_Prime[3] = np.interp(df_LI[1], df_LI_Filtering[5],
df_LI_Filtering[4])
        df_LI_Prime[4] = np.interp(df_LI[1], df_LI_Filtering[7],
df_LI_Filtering[6])
        df_LI_Prime[5] = np.interp(df_LI[1], df_LI_Filtering[9],
df_LI_Filtering[8])

    if Temporary_Rolling_Median_Factor != 0:
        print(Print_Format+"Preview Rolling Median applied")

    fig1, Main = plt.subplots(figsize=(3.3, 1.5),dpi=200)
```

```
    plt.subplots_adjust(left=0.080, right=0.710, bottom=0.090, top=1)
    Main.set_xlabel(TimeLabeling)
    Handles_Table = []

    if Number_Of_Columns == 4:
        Offset_a = Boxes_Status[1] * 60 - 60
        Offset_b = Offset_a + Boxes_Status[2] * 60
        if Import_Lab_RTD == 'y':
            Offset_c = Offset_b + Boxes_Status[6] * 60

        Main.set_ylabel(Labels[0])
        if Boxes_Status[0] == 0:
            p1, = Main.plot(df_LI_Prime[1] / Timescale, df_LI_Prime[0],
'#ffffff', linewidth=0.5)
            p1.axes.yaxis.set_visible(False)
        else:
            p1, = Main.plot(df_LI_Prime[1] / Timescale, df_LI_Prime[0],
'#ff940b', linewidth=0.5)
        Main.yaxis.label.set_color(p1.get_color())

        if Boxes_Status[0] == 0:
            Main.cla()
            Main.remove()

        if Boxes_Status[1] == 1:
            Demod_1_Volt = Main.twinx()
            Demod_1_Volt.set_ylabel(Labels[1])
            p2, = Demod_1_Volt.plot(df_LI_Prime[1] / Timescale,
df_LI_Prime[2]*1000,'#52DB00',linewidth=0.5)
            Demod_1_Volt.yaxis.label.set_color(p2.get_color())

        if Boxes_Status[2] == 1:
            Cell_Temperature = Main.twinx()
            Cell_Temperature.set_ylabel(Labels[2])
            p3, = Cell_Temperature.plot(df_MM[1] / Timescale, df_MM[2],
'green', linewidth=0.5)
            Cell_Temperature.yaxis.label.set_color(p3.get_color())
            Cell_Temperature.spines['right'].set_position(('outward',
Offset_b))

        if Import_Lab_RTD == 'y':
            if Boxes_Status[3] == 1:
                Lab_Temperature = Main.twinx()
                Lab_Temperature.set_ylabel(Labels[6])
                p4, = Lab_Temperature.plot(df_MM_RTD[1] / Timescale,
df_MM_RTD[0], '#9200DB', linewidth=0.5)
                Lab_Temperature.yaxis.label.set_color('#9200DB')
                Lab_Temperature.spines['right'].set_position(('outward',
Offset_c))

    if Number_Of_Columns >= 8:
        Offset_a = Boxes_Status[1] * 60 - 60
        Offset_b = Offset_a + Boxes_Status[2] * 60
        Offset_c = Offset_b + Boxes_Status[3] * 60
        Offset_d = Offset_c + Boxes_Status[4] * 60
        Offset_e = Offset_d + Boxes_Status[5] * 60
```

```
        Main.set_ylabel(Labels[0])
        if Boxes_Status[0] == 0:
            p1, = Main.plot(df_LI_Prime[1] / Timescale, df_LI_Prime[0],
'#ffffff', linewidth=0.5)
            p1.axes.yaxis.set_visible(False)
        else:
            p1, = Main.plot(df_LI_Prime[1] / Timescale, df_LI_Prime[0],
'#ff940b', linewidth=0.5)
        Main.yaxis.label.set_color(p1.get_color())

        if Boxes_Status[1] == 1:
            Demod_1_Volt = Main.twinx()
            Demod_1_Volt.set_ylabel(Labels[1])
            p2, = Demod_1_Volt.plot(df_LI_Prime[1] / Timescale,
df_LI_Prime[2] * 1000, '#52DB00', linewidth=0.5)
            Demod_1_Volt.yaxis.label.set_color(p2.get_color())

        if Boxes_Status[2] == 1:
            Demod_2_Freq = Main.twinx()
            Demod_2_Freq.set_ylabel(Labels[2])
            p3, = Demod_2_Freq.plot(df_LI_Prime[1]/Timescale,df_LI_Prime[3],
'#00C1FF', linewidth=0.5)
            Demod_2_Freq.yaxis.label.set_color(p3.get_color())
            Demod_2_Freq.spines['right'].set_position(('outward', Offset_b))

        if Boxes_Status[3] == 1:
            Demod_2_Volt = Main.twinx()
            Demod_2_Volt.set_ylabel(Labels[3])
            p4, =
Demod_2_Volt.plot(df_LI_Prime[1]/Timescale,df_LI_Prime[4]*1000, '#DC00FF',
linewidth=0.5)
            Demod_2_Volt.yaxis.label.set_color(p4.get_color())
            Demod_2_Volt.spines['right'].set_position(('outward', Offset_c))

        if Boxes_Status[4] == 1:
            Demod_3_Volt = Main.twinx()
            Demod_3_Volt.set_ylabel(Labels[4])
            p5, = Demod_3_Volt.plot(df_LI_Prime[1]/Timescale,df_LI_Prime[5],
'red', linewidth=0.5)
            Demod_3_Volt.yaxis.label.set_color(p5.get_color())
            Demod_3_Volt.spines['right'].set_position(('outward', Offset_d))

        if Boxes_Status[5] == 1:
            Cell_Temperature = Main.twinx()
            Cell_Temperature.set_ylabel(Labels[5])
            p6, = Cell_Temperature.plot(df_MM[1] / Timescale, df_MM[2],
'green', linewidth=0.5)
            Cell_Temperature.yaxis.label.set_color(p6.get_color())
            Cell_Temperature.spines['right'].set_position(('outward',
Offset_e))

        if Import_Lab_RTD == 'y':
            Offset_f = Offset_e + Boxes_Status[6] * 60
            if Boxes_Status[6] == 1:
                Lab_Temperature = Main.twinx()
                Lab_Temperature.set_ylabel(Labels[6])
                p7, = Lab_Temperature.plot(df_MM_RTD[1] / Timescale,
```

```
df_MM_RTD[0], '#9200DB', linewidth=0.5)
                Lab_Temperature.yaxis.label.set_color('#9200DB')
                Lab_Temperature.spines['right'].set_position(('outward',
Offset_f))

    Main.ticklabel_format(useOffset=False, style='plain')
    plt.show()
#================================================================================
=========================================#

#=================================== G - Peak Finder Algorithm Function
=========================================#
def Peak_Finder(df_Partial,t,Check_Compensation):
    Total_Time = df_Partial.iloc[-1][0]
    Timescale, Time_Label = Get_Time_Formating(Total_Time)

    Lab_Temperature = 20.5
    Peak_Delta = 2000
    Peak_Distance = Peak_Delta
    LI_Prominence = 0.15
    MM_Prominence = 0.001

    LI = df_Partial[1]
    MM = df_Partial[2]-273.15-Lab_Temperature

    Found_Peaks_LI = sign.find_peaks(LI,
height=np.min(LI),distance=Peak_Distance,prominence=LI_Prominence)
    Found_Height_LI = Found_Peaks_LI[1]['peak_heights']
    Found_Time_LI = t[Found_Peaks_LI[0]]

    Found_Peaks_MM = sign.find_peaks(MM,
height=np.min(MM),distance=Peak_Distance,prominence=MM_Prominence)
    Found_Height_MM = Found_Peaks_MM[1]['peak_heights']
    Found_Time_MM = t[Found_Peaks_MM[0]]

    Matching_Peaks = pd.DataFrame()
    Matching_Peak_Time_LI = []
    Matching_Peak_Time_MM = []
    Matching_Peak_Height_LI = []
    Matching_Peak_Height_MM = []

    for z in range(len(Found_Height_LI)):
        for x in range(len(Found_Height_MM)):
            if Check_Compensation == 0:
                if Found_Time_LI[z]-Found_Time_MM[x] < Peak_Delta and
Found_Time_LI[z]-Found_Time_MM[x] >= 0:
                    Matching_Peak_Time_LI.append(Found_Time_LI[z])
                    Matching_Peak_Time_MM.append(Found_Time_MM[x])
                    Matching_Peak_Height_LI.append(Found_Height_LI[z])
                    Matching_Peak_Height_MM.append(Found_Height_MM[x])
            elif Check_Compensation == 1:
                if np.abs(Found_Time_LI[z]-Found_Time_MM[x]) < Peak_Delta:
                    Matching_Peak_Time_LI.append(Found_Time_LI[z])
                    Matching_Peak_Time_MM.append(Found_Time_MM[x])
                    Matching_Peak_Height_LI.append(Found_Height_LI[z])
                    Matching_Peak_Height_MM.append(Found_Height_MM[x])
```

```
    Matching_Peaks.loc[:,0] = Matching_Peak_Time_LI
    Matching_Peaks.loc[:,1] = Matching_Peak_Time_MM
    Matching_Peaks.loc[:,2] = Matching_Peak_Height_LI
    Matching_Peaks.loc[:,3] = Matching_Peak_Height_MM

    print(Print_Format+'\n%d peaks were found for the Lock-In signal, while
%d peaks were found for the Multimeter signal. Between these, there are %d
matching
peaks.'%(len(Found_Time_LI),len(Found_Time_MM),len((Matching_Peak_Time_LI))))

    Peak_Time_Difference = Matching_Peaks.loc[:,0]-Matching_Peaks.loc[:,1]

    Mean_Peak_Time_Difference = (Peak_Time_Difference.mean())

    fig1, F = plt.subplots(figsize=(3.3, 1.5), dpi=300)
    plt.subplots_adjust(left=0.120, right=0.9, bottom=0.135, top=0.925)
    T_M = F.twinx()
    F.set_xlabel(Time_Label)
    F.set_ylabel("Resonator Frequency [Hz]")
    T_M.set_ylabel(r"$T_{Chamber}-20.5\ [\rm °C]$")
    p1, = F.plot(t/Timescale,LI,'#ff940b',linewidth=0.3)
    p2, = T_M.plot(t/Timescale,MM,'green', linewidth=0.3)
    F.scatter(Found_Time_LI/Timescale,Found_Height_LI, color='black', s=5,
marker='D', label='Unmatched Peaks')
    T_M.scatter(Found_Time_MM/Timescale,Found_Height_MM, color='black', s=5,
marker='D')
    F.scatter( Matching_Peaks.loc[:,0]/Timescale, Matching_Peaks.loc[:,2],
color='r', s=5, marker='D', label='Matched Peaks')
    T_M.scatter( Matching_Peaks.loc[:,1]/Timescale, Matching_Peaks.loc[:,3],
color='r', s=5, marker='D')
    plt.title('Lock-In | Multimeter Mean Lag = %f
s'%(Mean_Peak_Time_Difference))
    F.legend(loc='best',frameon=False)
    F.yaxis.label.set_color(p1.get_color())
    T_M.yaxis.label.set_color(p2.get_color())
    F.ticklabel_format(useOffset=False, style='plain')
    plt.show()

    return Mean_Peak_Time_Difference, df_Partial
#=========================================================================
========================================#

#============================================ 1 - Setting Text Formating
================================================#
Input_Format=Fore.GREEN+Style.BRIGHT
Warning_Format=Fore.RED+Style.BRIGHT
Print_Format=Fore.RESET+Style.BRIGHT
#=========================================================================
========================================#

#============================================ 2 - Setting Proprietary Path
================================================#
Folder_Path=input(Input_Format+"Input the Proprietary Path (See README): ")
#=========================================================================
========================================#

#============================================ 3 - Script Execution Option
```

```
=============================================#
Condition = 1
while Condition == 1:
    Script_Option = input(Input_Format+
                          '\nChoose desired option:\n          1) Generate
Merged Filtered Data or use Quick Plot GUI\n'
                          '          2) Load Pre-existing Merged Filtered
Data\nAnswer with corresponding number: ')
    if Script_Option == "1":
        Condition = 0
        Saved = 0
    elif Script_Option == "2":
        Condition = 0
        Saved = 1
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.")
#=============================================================================
=========================================#

#========================================== 4.1 - Merging Files In
Dataframes=========================================#
if Script_Option == "1":
    Condition = 1
    while Condition == 1:
        Import_Lab_RTD = input(Input_Format + "\nDoes this dataset have lab
temperature logging through RTD? [y/n] ")
        Import_Lab_RTD = Import_Lab_RTD.casefold()
        if Import_Lab_RTD == 'y':
            Condition = 0
        elif Import_Lab_RTD == 'n':
            Condition = 0
        else:
            print(Warning_Format+"\nInvalid answer. Please restart.")

    if Import_Lab_RTD == 'y':
        Number_Of_File_Type = 3
    else:
        Number_Of_File_Type = 2

    Number_Of_Files = int(len(glob.glob1(Folder_Path, "*.txt")) /
Number_Of_File_Type)

    i = 0
    index = str(i)
    df_LI = []
    df_MM = []
    df_MM_RTD = []

    print()
    while i <= Number_Of_Files-1:
        index = str(i)
        datalog = pd.read_csv(Folder_Path+'\LockIn_'+index+'.txt',sep=';',
skiprows=1, header=None)
        df_LI.append(datalog)
        datalog2 = pd.read_csv(Folder_Path+'\Multimeter_'+index+'.txt',
sep=';', skiprows=1, header=None)
        df_MM.append(datalog2)
```

```
        if i == 0:
            # open the project schedule
            Date_File = open(Folder_Path+'\LockIn_'+index+'.txt', 'r')
            Line_Text = Date_File.readline()
            Found_Date_Datetime_Format =
list(datefinder.find_dates(Line_Text))[0]
            Found_Date = str(Found_Date_Datetime_Format)

        if Import_Lab_RTD == 'y':
            datalog3 = pd.read_csv(Folder_Path + '\Multimeter_RTD_' + index +
'.txt', sep=';', skiprows=1, header=None)
            df_MM_RTD.append(datalog3)

        print(Print_Format+"File with index %d appended to dataframe" % (i))
        i = i+1

    df_LI=pd.concat(df_LI)
    df_MM=pd.concat(df_MM)
    if Import_Lab_RTD == 'y':
        df_MM_RTD = pd.concat(df_MM_RTD)
#=========================================================================
=======================================#


#============================================ 4.2 - Resistance Temperature
Conversion ==================================#
    if df_MM.iloc[0,0]<0:
        print(Print_Format+"\n4-Wire resistance values are negative. They
will be converted to positive values.")
        df_MM[0]=df_MM[0]*-1
    df_MM[0] = (1/298.15 + (1/3978) * np.log(df_MM[0]/1e4))**-1
    df_MM[2] = df_MM[0]-273.15

    if Import_Lab_RTD == 'y':
        a = 3.9083e-3
        b = -5.775e-7
        c = -4.183e-12
        R_Zero = 100.12
        df_MM_RTD[0] = (-a+np.sqrt(a**2-4*b*(1-(df_MM_RTD[0]/R_Zero))))/(2*b)
#=========================================================================
=======================================#


#============================================ 4.3 - Initializing Timestamps
=========================================#
    Number_Of_Columns=len(df_LI.columns)
    if Number_Of_Columns == 4:
        df_LI.iloc[0,1]=0
        df_LI.iloc[0,3]=0
    if Number_Of_Columns == 8:
        df_LI.iloc[0,1]=0
        df_LI.iloc[0,4]=0
        df_LI.iloc[0,7]=0
    if Number_Of_Columns >= 10:
        df_LI.iloc[0,1]=0
        df_LI.iloc[0,3]=0
```

```
        df_LI.iloc[0,5]=0
        df_LI.iloc[0,7]=0
        df_LI.iloc[0,9]=0
    print(Print_Format+"\nLock-In datalog has %d columns. Initial timestamps
have been set to 0."%(Number_Of_Columns))
#================================================================================
=========================================#

#========================================== 4.4 - Generate Quick Plot GUI
=========================================#
    global Rolling_Median_Factor
    global Trim_Start_Absolute
    global Trim_End_Absolute
    global Labels

    Rolling_Median_Factor = 0
    Trim_Start_Absolute = 0
    Trim_End_Absolute = 0

    window = Tk()
    window.title("Quick Plot GUI")
    window.geometry('1500x650')
    window['bg'] = '#373434'
    window.resizable(False, False)

    if Number_Of_Columns >= 2:
        Labels = ["Frequency [Hz]"]
        if Number_Of_Columns >= 4:
            Labels.append("Voltage [mV]")
            if Number_Of_Columns >= 6:
                if Number_Of_Columns >= 8:
                    Labels = ["Mode 1-1 Frequency [Hz]", "Mode 1-1 Voltage
[mV]", "Mode 1-2 Frequency [Hz]",
                              "Mode 1-2 Voltage [mV]", "Fringe Voltage [V]"]

    Labels.append('Cell TC Temperature [°C]')

    if Import_Lab_RTD == 'y':
        Labels.append('Lab RTD Temperature [°C]')

    dicts={}
    keys = range(len(Labels))
    values=[]

    Label(window, font=("Arial", "20", "bold","underline"), text="Plotted
Curves",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=50, y=5)

    Label(window, font=("Arial", "20", "bold","underline"), text="Timestamp
Type",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=500, y=5)

    Label(window, font=("Arial", "20", "bold","underline"), text="Trimming
```

```
Tool",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=1025, y=5)

    Label(window, font=("Arial", "14", "bold"), text="Input Beginning",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=950, y=100)

    Label(window, font=("Arial", "14", "bold"), text="Input End",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=950, y=150)

    Label(window, font=("Arial", "20", "bold","underline"), text="Rolling
Median Tool",
          bg='#373434', relief='solid',
          bd=3, fg='red', highlightbackground='black',
          width=20, height=2).place(x=1025, y=240)

    Time_Begin = DoubleVar()
    Entry(window, font=("Arial", "14", "bold"),textvariable=Time_Begin,
                      bg='#373434', relief='solid',
                      bd=3, fg='red', highlightbackground='black',
                      width=20).place(x=1225, y=110)

    Time_End = DoubleVar()
    Entry(window, font=("Arial", "14", "bold"),textvariable=Time_End,
                      bg='#373434', relief='solid',
                      bd=3, fg='red', highlightbackground='black',
                      width=20).place(x=1225, y=160)

    Rolling_Median = IntVar(window)
    Scale(window,from_=0,to=1000,length=342,resolution=10,
          orient='horizontal',font=("Arial", "14",
"bold"),tickinterval=250,showvalue=1,
          bg='#373434',troughcolor='#D99D08',relief='solid',bd=1,fg='red',
highlightbackground='black',
          variable=Rolling_Median).place(x=1025, y=340)

    Timestamp_Value = StringVar(window)
    for k in range(4):
        Text_Box=['Seconds [s]','Minutes [min]','Hours [hr]','Days [days]']
        Value_Box=['seconds','minutes','hours','days']
        Spacer=50*k
        Radiobutton(window, text=Text_Box[k],
                    bg='#373434', variable=Timestamp_Value,
value=Value_Box[k], command=partial(Plot_GUI_Update,dicts),
                    relief='solid', bd=3, fg='red',
highlightbackground='black', font=("Arial", "14", "bold"),
                    width=20, height=2).place(x=537, y=100+Spacer)

    for k in range(len(keys)):
        values.append(IntVar(window))
        dicts[keys[k]] = values[k]
        Spacer=50*k
```

```
        Checkbutton(window, text=Labels[k],
                    bg='#373434', variable=dicts[k], onvalue=1, offvalue=0,
command=partial(Plot_GUI_Update,dicts),
                    relief='solid', bd=3, fg='red',
highlightbackground='black', font=("Arial", "14", "bold"),
                    width=20, height=2).place(x=85, y=100+Spacer)

    Button(window, text='Plot',

command=partial(Plotting,df_LI,df_MM,df_MM_RTD,Number_Of_Columns,Labels,Impor
t_Lab_RTD), bg='#373434',
           relief='solid', bd=3, fg='green', highlightbackground='black',
font=("Arial", "14", "bold"),
           width=22, height=2).place(x=537, y=450)

    Button(window, text='Save',
           command=Save, bg='#373434',
           relief='solid', bd=3, fg='green', highlightbackground='black',
font=("Arial", "14", "bold"),
           width=22, height=2).place(x=537, y=500)

    Button(window, text='Exit',
           command=Exit, bg='#373434',
           relief='solid', bd=3, fg='green', highlightbackground='black',
font=("Arial", "14", "bold"),
           width=22, height=2).place(x=537, y=550)

    window.mainloop()
#=============================================================================
=======================================#

#========================================== 4.5 - Trimming and Rolling
Median ========================================#
    if Trim_Start_Absolute != 0:
        df_LI = df_LI.loc[df_LI[1] > Trim_Start_Absolute]
        df_MM = df_MM.loc[df_MM[1] > Trim_Start_Absolute]

        if Import_Lab_RTD == 'y':
            df_MM_RTD = df_MM_RTD.loc[df_MM_RTD[1] > Trim_Start_Absolute]


    if Trim_End_Absolute > Trim_Start_Absolute:
        df_LI = df_LI.loc[df_LI[1] < Trim_End_Absolute]
        df_MM = df_MM.loc[df_MM[1] < Trim_End_Absolute]

        if Import_Lab_RTD == 'y':
            df_MM_RTD = df_MM_RTD.loc[df_MM_RTD[1] < Trim_End_Absolute]

    df_LI_Filtering = pd.DataFrame()
    df_LI_Filtering = df_LI

    if Rolling_Median_Factor == 0:
        print(Print_Format+"\nRolling Median Factor: None")

    else:
        limit = 0.01
        df_LI_Filtering[Number_Of_Columns] =
```

```
df_LI_Filtering[0].rolling(Rolling_Median_Factor).median()
        df_LI_Filtering[Number_Of_Columns + 1] = df_LI_Filtering[0] -
df_LI_Filtering[Number_Of_Columns]
        df_LI_Filtering[Number_Of_Columns + 1] =
df_LI_Filtering[Number_Of_Columns + 1].abs()
        df_LI_Filtering =
df_LI_Filtering.drop(df_LI_Filtering[df_LI_Filtering[Number_Of_Columns + 1] >
limit].index)
        print(Print_Format+"\nRolling Median applied with Factor: %d" %
(Rolling_Median_Factor))

    if Trim_Start_Absolute == 0:
        print(Print_Format+'Trim Start: None')
    else:
        print(Print_Format+'Trim Start: %d s' % (Trim_Start_Absolute))

    if Trim_End_Absolute == 0:
        print(Print_Format+'Trim End: None')
    else:
        print(Print_Format+'Trim End: %d s' % (Trim_End_Absolute))
#================================================================================
=========================================#

#========================================== 4.6 - Data Filtering and
Timestamp Interpolation ========================#
    df_LI_Prime = pd.DataFrame()

    Final_Rate = 3

    if Number_Of_Columns == 4:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1]
        df_LI_Prime[2] = df_LI[2] = np.interp(df_LI[1], df_LI_Filtering[3],
df_LI_Filtering[2])

    if Number_Of_Columns == 8:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1].to_numpy()
        df_LI_Prime[2] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[2])
        df_LI_Prime[3] = np.interp(df_LI[1], df_LI_Filtering[4],
df_LI_Filtering[3])
        df_LI_Prime[4] = np.interp(df_LI[1], df_LI_Filtering[4],
df_LI_Filtering[5])
        df_LI_Prime[5] = np.interp(df_LI[1], df_LI_Filtering[7],
df_LI_Filtering[6])

    if Number_Of_Columns == 10:
        df_LI_Prime[0] = np.interp(df_LI[1], df_LI_Filtering[1],
df_LI_Filtering[0])
        df_LI_Prime[1] = df_LI[1].to_numpy()
        df_LI_Prime[2] = np.interp(df_LI[1], df_LI_Filtering[3],
df_LI_Filtering[2])
        df_LI_Prime[3] = np.interp(df_LI[1], df_LI_Filtering[5],
df_LI_Filtering[4])
```

```
        df_LI_Prime[4] = np.interp(df_LI[1], df_LI_Filtering[7],
df_LI_Filtering[6])
        df_LI_Prime[5] = np.interp(df_LI[1], df_LI_Filtering[9],
df_LI_Filtering[8])

    df_Total = pd.DataFrame()

    MM_Length = len(df_MM.index)
    MM_Beginning_Time = df_MM.iloc[0][1]
    MM_End_Time = df_MM.iloc[-1][1]
    MM_Streaming_Rate = MM_Length/(MM_End_Time-MM_Beginning_Time)
    MM_sos = sign.butter(N=4, Wn=Final_Rate, fs=MM_Streaming_Rate,
output='sos')

    LI_Length = len(df_LI_Prime.index)
    LI_Beginning_Time = df_LI_Prime.iloc[0][1]
    LI_End_Time = df_LI_Prime.iloc[-1][1]
    LI_Streaming_Rate = LI_Length / (LI_End_Time - LI_Beginning_Time)
    LI_sos = sign.butter(N=4, Wn=Final_Rate, fs=LI_Streaming_Rate,
output='sos')

    Interp_Time = np.linspace(int((LI_Beginning_Time + MM_Beginning_Time)/2),
int((LI_End_Time + MM_End_Time)/2),
                              int(Final_Rate*(LI_End_Time + MM_End_Time)/2))

    if Trim_Start_Absolute != 0:

Found_Date_Datetime_Format=Found_Date_Datetime_Format+timedelta(seconds=Inter
p_Time[0])
        Found_Date=str(Found_Date_Datetime_Format)
        df_Total[0] = Interp_Time-Interp_Time[0]

    else:
        df_Total[0] = Interp_Time

    MM_Time = df_MM.iloc[:,1]
    MM_Time = sign.sosfiltfilt(MM_sos, MM_Time)

    MM_Temperature = df_MM.iloc[:,0]
    MM_Temperature = sign.sosfiltfilt(MM_sos, MM_Temperature)
    MM_Interp = np.interp(Interp_Time, MM_Time, MM_Temperature)

    if Number_Of_Columns >= 2:
        LI_Time = df_LI_Prime.iloc[:, 1]
        LI_Time = sign.sosfiltfilt(LI_sos, LI_Time)

        LI_F_1 = df_LI_Prime.iloc[:, 0]
        LI_F_1 = sign.sosfiltfilt(LI_sos, LI_F_1)
        LI_F_1_Interp = np.interp(Interp_Time, LI_Time, LI_F_1)
        df_Total[1] = LI_F_1_Interp
        df_Total[2] = MM_Interp

        if Number_Of_Columns >= 4:
            LI_V_1 = df_LI_Prime.iloc[:, 2]
            LI_V_1 = sign.sosfiltfilt(LI_sos, LI_V_1)
            LI_V_1_Interp = np.interp(Interp_Time, LI_Time, LI_V_1)
            df_Total[3] = LI_V_1_Interp
```

```
            if Number_Of_Columns == 8:
                LI_F_2 = df_LI_Prime.iloc[:, 3]
                LI_F_2 = sign.sosfiltfilt(LI_sos, LI_F_2)
                LI_F_2_Interp = np.interp(Interp_Time, LI_Time, LI_F_2)
                df_Total[4] = LI_F_2_Interp

                LI_V_2 = df_LI_Prime.iloc[:, 4]
                LI_V_2 = sign.sosfiltfilt(LI_sos, LI_V_2)
                LI_V_2_Interp = np.interp(Interp_Time, LI_Time, LI_V_2)
                df_Total[5] = LI_V_2_Interp

                LI_V_3 = df_LI_Prime.iloc[:, 5]
                LI_V_3 = sign.sosfiltfilt(LI_sos, LI_V_3)
                LI_V_3_Interp = np.interp(Interp_Time, LI_Time, LI_V_3)
                df_Total[6] = LI_V_3_Interp

            if Number_Of_Columns == 10:
                LI_F_2 = df_LI_Prime.iloc[:, 3]
                LI_F_2 = sign.sosfiltfilt(LI_sos, LI_F_2)
                LI_F_2_Interp = np.interp(Interp_Time, LI_Time, LI_F_2)
                df_Total[4] = LI_F_2_Interp

                LI_V_2 = df_LI_Prime.iloc[:, 4]
                LI_V_2 = sign.sosfiltfilt(LI_sos, LI_V_2)
                LI_V_2_Interp = np.interp(Interp_Time, LI_Time, LI_V_2)
                df_Total[5] = LI_V_2_Interp

                LI_V_3 = df_LI_Prime.iloc[:, 5]
                LI_V_3 = sign.sosfiltfilt(LI_sos, LI_V_3)
                LI_V_3_Interp = np.interp(Interp_Time, LI_Time, LI_V_3)
                df_Total[6] = LI_V_3_Interp
#================================================================================
=========================================#

#============================================ 4.7 - Merged Dataset Saving
=========================================#
    Condition = 1
    while Condition == 1:
        Save_Merged_Filtered_Df = input(Input_Format + "\nWould you like to
save the merged and filtered dataframe? [y/n] ")
        Save_Merged_Filtered_Df = Save_Merged_Filtered_Df.casefold()
        if Save_Merged_Filtered_Df == 'y':
            Condition = 0
        elif Save_Merged_Filtered_Df == 'n':
            Condition = 0
        else:
            print(Warning_Format+"\nInvalid answer. Please restart.")

    S=" ; "
    if Number_Of_Columns >= 2:
        File_Header="Time [s]"+S+"Frequency [Hz]"+S+"Multimeter Temperature
[K]"+S+"Rolling_Median_Factor = "+str(Rolling_Median_Factor)
        if Number_Of_Columns >= 4:
            File_Header="Time [s]"+S+"Frequency [Hz]"+S+"Multimeter
Temperature [K]"+S+"Voltage [V]"+S+"Rolling_Median_Factor =
"+str(Rolling_Median_Factor)
```

```
            if Number_Of_Columns >= 6:
                if Number_Of_Columns >= 8:
                    File_Header = "Time [s]"+S+"Mode 1-1 Frequency
[Hz]"+S+"Multimeter Temperature [K]"+S+"Mode 1-1 Voltage [V]"+S+"Mode 1-2
Frequency [Hz]"+S+"Mode 1-2 Voltage [V]"+S+"Fringe Voltage [V]"

    if Save_Merged_Filtered_Df == "y":
        Saved = 1
        np.savetxt(Folder_Path + "\Datasets\Merged_Filtered_Data.txt",
                   df_Total,
                   delimiter=';',
                   header= File_Header+S+"Rolling_Median_Factor =
"+str(Rolling_Median_Factor)+S+'Trim Start [s]:
'+str(Trim_Start_Absolute)+S+'Trim End [s]:
'+str(Trim_End_Absolute)+S+Found_Date,
                   comments='', fmt='%s')
        print(Print_Format+"\nThe merged and filtered dataframe has been
saved at this location:
%s"%(Folder_Path+"\Datasets\Merged_Filtered_Data.txt"))
    Script_Option="2"

#========================================== 5.1 - Time-Lag Compensation and
Lag Compensated Dataset Saving ==========#
if Script_Option == "2":
    df_Merged = pd.DataFrame()
    if Saved == 1:
        df_Merged = pd.read_csv(Folder_Path +
'\Datasets\Merged_Filtered_Data.txt', sep=';', skiprows=1, header=None)
        with open(Folder_Path + '\Datasets\Merged_Filtered_Data.txt') as
header_of_file:
            Dataset_Header = header_of_file.readline().strip('\n')
    elif Saved == 0:
        df_Merged = df_Total
        Dataset_Header=File_Header+S+"Rolling_Median_Factor =
"+str(Rolling_Median_Factor)+S+'Trim Start [s]:
'+str(Trim_Start_Absolute)+S+'Trim End [s]:
'+str(Trim_End_Absolute)+S+Found_Date

    Number_Of_Columns = len(df_Merged.columns)

    Condition = 1
    while Condition == 1:
        Time_Compensate = input(Input_Format + "\nWould you like to execute
the time compensation algorithm? [y/n] ")
        Time_Compensate = Time_Compensate.casefold()
        if Time_Compensate == 'y':
            Condition = 0
        elif Time_Compensate == 'n':
            Condition = 0
        else:
            print(Warning_Format+"\nInvalid answer. Please restart.")
    if Time_Compensate == "y":

        Date_File = open(Folder_Path + '\Datasets\Merged_Filtered_Data.txt',
'r')
        Line_Text = Date_File.readline()
        Found_Date_Datetime_Format = list(datefinder.find_dates(Line_Text))[-
```

```
1]

        Time_Partial = df_Merged[0].to_numpy()
        Time_Partial_Offsetted = Time_Partial

        Check_Compensation = 0

Mean_Peak_Time_Difference,df_Merged=Peak_Finder(df_Merged,Time_Partial,Check_
Compensation)
        Uncompensated_Lag_Header = "Uncompensated Lag = " +
str(Mean_Peak_Time_Difference)+' s'

        Time_Partial_Offsetted=Time_Partial+Mean_Peak_Time_Difference
        Original_Mean_Peak_Time_Difference=Mean_Peak_Time_Difference

        df_Merged[2] =
np.interp(Time_Partial,Time_Partial_Offsetted,df_Merged[2])

        Check_Compensation = 1

Mean_Peak_Time_Difference,df_Merged=Peak_Finder(df_Merged,Time_Partial,Check_
Compensation)
        Found_Date = str(Found_Date_Datetime_Format)

        df_Merged = df_Merged.loc[df_Merged[0] >
Original_Mean_Peak_Time_Difference]
        Found_Date_Datetime_Format = Found_Date_Datetime_Format +
timedelta(seconds=df_Merged.iloc[0,0])
        Found_Date=str(Found_Date_Datetime_Format)

        pd.options.mode.chained_assignment = None
        df_Merged.loc[:,0] -= df_Merged.iloc[0,0]
        pd.options.mode.chained_assignment = 'warn'

        Compensated_Lag_Header = "Compensated Lag =
"+str(Mean_Peak_Time_Difference)+' s'

        Condition = 1
        while Condition == 1:
            Save_Merged_Filtered_Df_Time_Compensated = input(Input_Format +
"\nWould you like to save the time compensated merged and filtered dataframe?
[y/n] ")
            Save_Merged_Filtered_Df_Time_Compensated =
Save_Merged_Filtered_Df_Time_Compensated.casefold()
            if Save_Merged_Filtered_Df_Time_Compensated == 'y':
                Condition = 0
            elif Save_Merged_Filtered_Df_Time_Compensated == 'n':
                Condition = 0
            else:
                print(Warning_Format+"\nInvalid answer. Please restart.")
        if Save_Merged_Filtered_Df_Time_Compensated == "y":
            S = " ; "


np.savetxt(Folder_Path+"\Datasets\Merged_Filtered_Data_Time_Compensated.txt",
df_Merged,delimiter=';',
                header=Dataset_Header+S+'Compensated Date:
```

```
'+Found_Date+S+Uncompensated_Lag_Header+S+Compensated_Lag_Header,
               comments='', fmt='%s')
            print(Print_Format+"\nThe merged and filtered dataframe has been
saved at this location: %s" %
(Folder_Path+"\Datasets\Merged_Filtered_Data_Time_Compensated.txt"))
#================================================================================
=========================================#
```

Colourmap Plotter.py

```
#============================================== A - Importing Libraries
==============================================#
print("\nRead 'Color Map Plotter README.txt' for any information about this
script.")
print('\nLoading Libraries...')
import os
import datefinder
import numpy as np
import pandas as pd
import matplotlib.ticker
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from colorama import Fore, Back, Style
from matplotlib.offsetbox import VPacker
print('\nLibraries Loaded!\n')
#============================================================================
=========================================#

#========================================== B - Offset Formating Function
==============================================#
class OffsetGet(matplotlib.ticker.ScalarFormatter):
    def get_offset(self):
        if len(self.locs) == 0:
            return ''
        return self.offset
#============================================================================
=========================================#

#======================================== C - Quick Plot Dimension Modifier
==============================================#
def Plot_Settings_Modifier(Which_Plot):
    if Which_Plot == "Raw_Data_Plot":
        P_S = {'x_dim': 3, 'y_dim': 1.5, 'DPI': 300, 'left': 0.093, 'right':
0.807, 'bottom': 0.126, 'top': 0.955}
    elif Which_Plot == "Raw_Data_Temperature_Equivalence":
        P_S = {'x_dim': 3.3, 'y_dim': 1.5, 'DPI': 300, 'left': 0.120,
'right': 0.998, 'bottom': 0.135, 'top': 0.998}
    elif Which_Plot == "Colormap_Plot":
        P_S = {'x_dim': 3.6, 'y_dim': 2.75, 'DPI': 300, 'left': 0.140,
'right': 0.938, 'bottom': 0.157, 'top': 0.979}
    return P_S
#============================================================================
=========================================#

#========================================== D - Data Conversion Function
==============================================#
def Conversion(df_Partial,T_Zero,F_Zero, prop, prms):
    T_MM_Relative_Lab=df_Partial[2]-(273.15)-(prms['Lab_Temperature'])
    Delta_T_MM = df_Partial[2]-T_Zero
    Delta_T_LI = ((df_Partial[1]-
F_Zero)/F_Zero)*2*(prop['Sigma_Zero']/(prop['E_Prime_SiN']*(prop['CTE_Si']-
prop['CTE_SiN'])))
    return T_MM_Relative_Lab, Delta_T_MM, Delta_T_LI
```

```
#=============================================================================
=========================================#

#============================================ E - Raw Data Plot Function
=========================================#
def Plotting(Frequency_LI, Temp_Rel_Lab_MM, Frequency_LI_2, Time_Partial,
prms):
    plt.rcParams.update({'font.size': 9})

    Condition = 1
    while Condition == 1:
        Mode_1_2_Plotting = input(Input_Format+"Would you like plot Mode 1-2
as well? [y/n] ")
        Mode_1_2_Plotting = Mode_1_2_Plotting.casefold()
        if Mode_1_2_Plotting == 'y':
            Condition = 0
        elif Mode_1_2_Plotting == 'n':
            Condition = 0
        else:
            print(Warning_Format+"\nInvalid answer. Please restart.")

    Which_Plot = "Raw_Data_Plot"
    P_S=Plot_Settings_Modifier(Which_Plot)
    fig1, F = plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']),
dpi=P_S['DPI'])
    plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
    T_M = F.twinx()

    if Mode_1_2_Plotting == 'y':
        F2 = F.twinx()

    Mean_Freq = []
    Mean_Freq2 = []

    for m in range(Number_Of_Folders):
        T_R_L_MM = Temp_Rel_Lab_MM[m]
        Time = Time_Partial[m]
        Freq = Frequency_LI[m]
        Mean_Freq = np.append(Mean_Freq, np.mean(Freq))

        if Mode_1_2_Plotting == 'y':
            Freq2 = Frequency_LI_2[m]
            Mean_Freq2 = np.append(Mean_Freq2, np.mean(Freq2))

        p1, = F.plot(Time/prms['Timescale'], Freq, 'black', linewidth=0.3,
label="Mode 1-1 Frequency [Hz]")
        F.tick_params(axis='y', colors=p1.get_color())

        p2, = T_M.plot(Time/prms['Timescale'], T_R_L_MM, 'green',
linewidth=0.3, label=r"$T_{Cell}-20.5\ [\rm °C]$")
        T_M.tick_params(axis='y', colors=p2.get_color())

        if Mode_1_2_Plotting == 'y':
            p3, = F2.plot(Time/prms['Timescale'], Freq2, '#00C1FF',
linewidth=0.3, label="Mode 1-2 Frequency [Hz]")
            F2.tick_params(axis='y', colors=p3.get_color())
```

```
    plt.draw()

    F.set_xlabel(prms['Time_Label'])
    F.set_ylabel("Mode 1-1 Frequency [Hz]")
    F.yaxis.label.set_color(p1.get_color())
    F.yaxis.set_major_formatter(OffsetGet())

F.ticklabel_format(axis='y',useOffset=float("{:.0f}".format(np.mean(Mean_Freq
))), style='plain')
    T_M.yaxis.label.set_color(p2.get_color())
    T_M.set_ylabel(r"$T_{Cell}-20.5\ [\rm °C]$")

    custom_lines = [Line2D([0], [0], color='black', lw=1),
                    Line2D([0], [0], color='green', lw=1)]
    F.legend(custom_lines, ['Mode 1-1 Frequency [Hz]', 'Thermistor
Temperature'], loc='upper left', frameon=False,
             prop={'size': 6})

    if Mode_1_2_Plotting == 'y':
        F2.yaxis.label.set_color(p3.get_color())
        F2.set_ylabel("Mode 1-2 Frequency [Hz]")
        F2.spines['right'].set_position(('outward', 45))
        F2.yaxis.set_major_formatter(OffsetGet())
        F2.ticklabel_format(axis='y',
useOffset=float("{:.0f}".format(np.mean(Mean_Freq2))), style='plain')
        custom_lines = [Line2D([0], [0], color='black', lw=1),
                        Line2D([0], [0], color='green', lw=1),
                        Line2D([0], [0], color='#00C1FF', lw=1)]
        F.legend(custom_lines, ['Mode 1-1 Frequency [Hz]', 'Thermistor
Temperature','Mode 1-2 Frequency [Hz]'],
                 loc='upper left', frameon=False, prop={'size': 6})

    plt.show()
#=============================================================================
=========================================#

#==================================== F - Temperature Equivalence Plot
Function ======================================#
def Plotting_2(Temp_LI, Temp_MM, Time_Partial, prms):
    Which_Plot = "Raw_Data_Temperature_Equivalence"
    P_S=Plot_Settings_Modifier(Which_Plot)
    plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']), dpi=P_S['DPI'])
    plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
    plt.xlabel(prms['Time_Label'])
    plt.ylabel("ΔT [K]")
    for m in range(Number_Of_Folders):
        D_T_LI = Temp_LI[m]
        D_T_MM = Temp_MM[m]
        Time = Time_Partial[m]
        p1, = plt.plot(Time/prms['Timescale'], D_T_LI, '#ff940b',
linewidth=0.3, label='Resonator Frequency')
        p2, = plt.plot(Time/prms['Timescale'], D_T_MM, 'green',
linewidth=0.3, label='Thermistor Temperature')
        plt.draw()
    leg = plt.legend(handles=[p1, p2], loc='upper left', frameon=False)
```

```
    for legobj in leg.legendHandles:
        legobj.set_linewidth(1.5)
    plt.show()
#=========================================================================
========================================#

#========================================== G - Colormap Plot Function
==========================================#
def Plotting_3(df_Total, prms):
    print(Print_Format+"\nStarting to generate color map plot. Process can be
very long and cause the program to crash.")
    T_M=df_Total[2]-(273.15)-(prms['Lab_Temperature'])
    matplotlib.offsetbox.VPacker()
    plt.rcParams.update({'font.size': 9})
    Which_Plot = "Colormap_Plot"
    P_S = Plot_Settings_Modifier(Which_Plot)
    plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']), dpi=P_S['DPI'])
    plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])

plt.scatter(T_M,df_Total[1],cmap="gnuplot",c=df_Total[0]/prms['Seconds_To_Day
s'],marker="D",s=0.001)
    plt.ylabel("Frequency [Hz]")
    plt.xlabel(r"$T_{Cell}-20.5\ [\rm °C]$")

plt.gca().xaxis.set_major_formatter(matplotlib.ticker.StrMethodFormatter('{x:
,.2f}'))
    plt.gca().yaxis.set_major_formatter(OffsetGet())
    Mean_Frequency = np.mean(df_Total[1].to_numpy())
    Max_Frequency = np.max(df_Total[1].to_numpy())
    Min_Frequency = np.min(df_Total[1].to_numpy())
    plt.ticklabel_format(axis='y',
useOffset=float("{:.2f}".format(Mean_Frequency)), style='plain')
    Tick_Lowest = (Mean_Frequency-(Mean_Frequency-Min_Frequency)*0.9)
    Tick_Lower = (Mean_Frequency-(Mean_Frequency-Min_Frequency)*0.45)
    Tick_Middle = Mean_Frequency
    Tick_Higher = (Max_Frequency-Mean_Frequency)*0.45+Mean_Frequency
    Tick_Highest = (Max_Frequency-Mean_Frequency)*0.9+Mean_Frequency
    plt.yticks([Tick_Middle-10,Tick_Middle-
5,Tick_Middle,Tick_Middle+5,Tick_Middle+10],['-10','','0','','10'])
    Mean_Days = np.mean(df_Total[0].to_numpy())/prms['Seconds_To_Days']
    Max_Days = np.max(df_Total[0].to_numpy())/prms['Seconds_To_Days']
    Min_Days = np.min(df_Total[0].to_numpy())/prms['Seconds_To_Days']
    Tick_Lowest_Time = Min_Days
    Tick_Lower_Time = Mean_Days-(Mean_Days-Min_Days)*0.5
    Tick_Middle_Time = Mean_Days
    Tick_Higher_Time = (Max_Days-Mean_Days)*0.5+Mean_Days
    Tick_Highest_Time = Max_Days
    Color_Map=plt.colorbar()

Color_Map.set_ticks([Tick_Lowest_Time,Tick_Lower_Time,Tick_Middle_Time,Tick_H
igher_Time,Tick_Highest_Time])

Color_Map.set_ticklabels(['0','',int(Tick_Middle_Time),'',int(Tick_Highest_Ti
me)])
    Color_Map.set_label(prms['Time_Label'])
```

```
    plt.text(2.7,Mean_Frequency+9, r'+%d
Hz'%(Mean_Frequency),verticalalignment='top',
horizontalalignment='left',fontsize=9)

    plt.show()
#=============================================================================
=========================================#

#============================================ 1 - Setting Text Formating
=========================================#
Input_Format=Fore.GREEN+Style.BRIGHT
Warning_Format=Fore.RED+Style.BRIGHT
Print_Format=Fore.RESET+Style.BRIGHT
#=============================================================================
=========================================#

#============================================ 2 - Material Properties
=========================================#
ESiN = 250000
#Silicon nitride modulus of elasticity [MPa]
VSiN = 0.23
#Silicon nitride poisson's ratio
CTE_Si = 2.6e-6
#Silicon CTE
CTE_SiN = 2.3e-6
#Silicon nitride CTE
E_Prime_SiN = ESiN/(1-VSiN)
#Modified modulus of elasticity [MPa]
Sigma_Zero = 100
#Silicon nitride initial stress [MPa]
dct_prop =
{'CTE_Si':CTE_Si,'CTE_SiN':CTE_SiN,'E_Prime_SiN':E_Prime_SiN,'Sigma_Zero':Sig
ma_Zero}
#=============================================================================
=========================================#

#============================================ 3 - Data Processing Parameters
=========================================#
Drive_Label = 'A'
Final_Rate = 3
Lab_Temperature = 20.5
Seconds_To_Days = 60*60*24
dct_para =
{'Final_Rate':Final_Rate,'Lab_Temperature':Lab_Temperature,'Seconds_To_Days':
Seconds_To_Days}
#=============================================================================
=========================================#

#============================================ 4 - Choice of Data to Load
=========================================#
Condition = 1
while Condition == 1:
    Path_Choice = int(input(Input_Format+'Which datalog do you wish to
analyse:\n          1) 1 Mode With Vacuum Leak\n          2) 2 Modes Good
Aging Data\n          3) 2 Modes Outside Fridge\n          4) Single Dataset
Test Sample\nAnswer with either 1, 2, 3 or 4: '))
    if Path_Choice == 1:
```

```
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\1 Mode With Vacuum Leak'
        Condition = 0
    elif Path_Choice == 2:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\2 Modes Good Aging Data'
        Condition = 0
    elif Path_Choice == 3:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\2 Modes Outside Fridge'
        Condition = 0
    elif Path_Choice == 4:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\Single Dataset Test Sample'
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.\n")

Condition = 1
while Condition == 1:
    File_To_Use = int(input(Input_Format+'\nWould you liked to use:\n
1) Merged Filtered Data\n          2) Merged Filtered Data Time
Compensated\nAnswer with either 1 or 2: '))
    if File_To_Use == 1:
        File_Termination = "\Datasets\Merged_Filtered_Data.txt"
        Condition = 0
    elif File_To_Use == 2:
        File_Termination =
"\Datasets\Merged_Filtered_Data_Time_Compensated.txt"
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.\n")
#=============================================================================
=========================================#

#============================== 5 - Loading Data, Conversion & Large
Dataframe Creation ==============================#
Filepath=os.listdir(Drive_Label+Mainpath)
Number_Of_Folders = len(Filepath)

j = 0

df_Total = []
Temp_MM=[]
Temp_LI=[]
Time_Partial=[]
Frequency_LI=[]
Frequency_LI_2=[]
Temp_Rel_Lab_MM=[]
Date_Array_Start = []
Date_Array_Duration = []
Time_Gap_Array = []
```

```
while j <= Number_Of_Folders - 1:
    print(Print_Format+"\nWorking on storing folder with index %d in a
dataframe" % (j))
    i = 0
    index = str(i)
    df_Partial = pd.DataFrame()
    df_Partial =
pd.read_csv(Drive_Label+Mainpath+'\\'+Filepath[j]+'\Datafiles'+File_Terminati
on, sep=';', skiprows=1, header=None)

    Date_File =
open(Drive_Label+Mainpath+'\\'+Filepath[j]+'\Datafiles'+File_Termination,
'r')
    Line_Text = Date_File.readline()
    Found_Date = list(datefinder.find_dates(Line_Text))
    if len(Found_Date) > 1:
        Found_Date = str(Found_Date[2])
    else:
        Found_Date = str(Found_Date[1])
    Stripped_Date=datetime.strptime(Found_Date,"%Y-%m-%d %H:%M:%S.%f")
    Date_Array_Start.append(Stripped_Date)
    Date_Array_Duration.append(df_Partial.iloc[-1][0])

    if j != 0:
        Time_Gap_Array.append((Date_Array_Start[j]-Date_Array_Start[j-
1]).total_seconds()-Date_Array_Duration[j-1])
        df_Partial[0]=df_Partial[0]+Time_Gap_Array[j-1]+Final_Time

    print(Print_Format+"Folder with index %d stored in a dataframe" % (j))
    if j == 0:
        T_Zero = df_Partial.iloc[0][2]
        F_Zero = df_Partial.iloc[0][1]
    Final_Time = df_Partial.iloc[-1][0]

    T_MM_Relative_Lab, Delta_T_MM, Delta_T_LI = Conversion(df_Partial,
T_Zero, F_Zero, dct_prop, dct_para)

    Frequency_LI.append(df_Partial[1].to_numpy())
    Frequency_LI_2.append(df_Partial[4].to_numpy())
    Temp_LI.append(Delta_T_LI)
    Temp_Rel_Lab_MM.append(T_MM_Relative_Lab)
    Temp_MM.append(Delta_T_MM)
    Time_Partial.append(df_Partial[0].to_numpy())

    df_Total.append(df_Partial)

    j+=1

df_Total = pd.concat(df_Total)

print(Print_Format+"\nAll of the datalogs are now fully merged, filtered and
interpolated to a rate of %d Hz\n"%(Final_Rate))
print(Print_Format+"The total duration of the analysed data is of %f
days\n"%(float(df_Total.iloc[-1][0])/Seconds_To_Days))
#=============================================================================
=========================================#
```

```
#======================================== 6 - Timescale & Time Label Finder
========================================#
Total_Time = df_Total.iloc[-1][0]
if Total_Time < 3600:
    Timescale = 1
    Time_Label = 'Time [Seconds]'
elif Total_Time < 18000:
    Timescale = 60
    Time_Label = 'Time [Minutes]'
elif Total_Time < 259200:
    Timescale = 60 * 60
    Time_Label = 'Time [Hours]'
else:
    Timescale = 60 * 60 * 24
    Time_Label = 'Time [Days]'
dct_para['Timescale']=Timescale
dct_para['Time_Label']=Time_Label
#================================================================================
========================================#

#========================================== 7 - Raw Data Plot Function
==========================================#
Plotting(Frequency_LI, Temp_Rel_Lab_MM, Frequency_LI_2, Time_Partial,
dct_para)
#================================================================================
========================================#

#==================================== 8 - Temperature Equivalence Plot
Function ====================================#
Plotting_2(Temp_LI, Temp_MM, Time_Partial, dct_para)
#================================================================================
========================================#

#========================================== 9 - Colorplot Function
==========================================#
Plotting_3(df_Total, dct_para)
#================================================================================
========================================#
```

Frequency Drift Plotter.py

```
#============================================== A - Importing Libraries
================================================#
print("\nRead 'Frequency Drift Plotter README.txt' for any information about
this script.")
print('\nLoading Libraries...')
import os
import datefinder
import numpy as np
import pandas as pd
import scipy.signal as sign
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from scipy.optimize import curve_fit
from colorama import Fore, Back, Style
from matplotlib.ticker import ScalarFormatter
from matplotlib.ticker import StrMethodFormatter
print('\nLibraries Loaded!\n')
#=============================================================================
=========================================#

#=========================================== B - Quick Plot Dimension Modifier
=========================================#
def Plot_Settings_Modifier(Which_Plot):
    if Which_Plot == 'Partial_Drift_Plot':
        P_S = {'x_dim': 3.3, 'y_dim': 1.5, 'DPI': 300, 'left': 0.226,
'right': 0.929, 'bottom': 0.286, 'top': 0.890}
    elif Which_Plot == 'Partial_Drift_Plot':
        P_S = {'x_dim': 3.3, 'y_dim': 1.5, 'DPI': 300, 'left': 0.226,
'right': 0.929, 'bottom': 0.286, 'top': 0.890}
    elif Which_Plot == 'Test_Drift_Trend':
        P_S = {'x_dim': 3.5, 'y_dim': 1.24, 'DPI': 300, 'left': 0.180,
'right': 0.998, 'bottom': 0.32, 'top': 0.998}
    elif Which_Plot == 'Drift_Plot_With_Trend':
        P_S = {'x_dim': 3, 'y_dim': 1.24, 'DPI': 300, 'left': 0.180, 'right':
0.998, 'bottom': 0.32, 'top': 0.998}
    return P_S
#=============================================================================
=========================================#

#=========================================== C - Timescale & Time Label Finder
=========================================#
def Time_Scaler(Total_Time):
    if Total_Time < 3600:
        Timescale = 1
        Time_Label = 'Time [Seconds]'
    elif Total_Time < 18000:
        Timescale = 60
        Time_Label = 'Time [Minutes]'
    elif Total_Time < 259200:
        Timescale = 60 * 60
        Time_Label = 'Time [Hours]'
    else:
        Timescale = 60 * 60 * 24
```

```
        Time_Label = 'Time [Days]'
    return Timescale,Time_Label
#=============================================================================
=========================================#

#====================================== D - Double Logarithmic Trend
Function ========================================#
def Log_Log_Fit(t,A1,A2,t0a,B1,B2,t0b):
    return A1*np.log(A2*t+t0a)+B1*np.log(B2*t+t0b)
#=============================================================================
=========================================#

#====================================== E - Logarithmic Linear Trend
Function ========================================#
def Log_Lin_Fit(t,A1,A2,t0a,B1,t0b):
    return A1*np.log(A2*t+t0a)+B1*t+t0b
#=============================================================================
=========================================#

#=================== F - Temperature Compensation Trend Generator &
Individual Colormap Plotter ====================#
def Plotting(m,prms,df_Partial,T_MM_Relative_Lab,Individual_Plotting):
    Slope_Fit = []
    Offset_Fit = []
    Fitting = []
    Slope_Fit_1_2 = []
    Offset_Fit_1_2 = []
    Fitting_1_2 = []

    T_R_L_MM=T_MM_Relative_Lab
    Time=df_Partial[0]
    Freq=df_Partial[1]

    z=0

    Start = df_Partial.iloc[-1][0]
    Timescale,Time_Label = Time_Scaler(Start)
    Multiples = Start / (prms['Averaging_Interval_Seconds'])

    while z < int(Multiples):
        Lower_Border = z * (prms['Averaging_Interval_Seconds'])
        Upper_Border = (z + 1) * (prms['Averaging_Interval_Seconds'])

        df_Sectionned = pd.DataFrame()
        df_Sectionned = df_Partial.loc[df_Partial[0] > Lower_Border]
        df_Sectionned = df_Sectionned.loc[df_Sectionned[0] < Upper_Border]

        Partial_Fitting = np.polyfit(df_Sectionned[2]-273.15-
prms['Lab_Temperature'], df_Sectionned[1], deg=1)
        Partial_Fitting_1_2 = np.polyfit(df_Sectionned[2]-273.15-
prms['Lab_Temperature'], df_Sectionned[4], deg=1)

        Slope_Fit.append(Partial_Fitting[0])
        Offset_Fit.append(Partial_Fitting[1])

        Slope_Fit_1_2.append(Partial_Fitting_1_2[0])
        Offset_Fit_1_2.append(Partial_Fitting_1_2[1])
```

```
        z+=1

    Fitting.append(np.mean(Slope_Fit))
    Fitting.append(np.mean(Offset_Fit))

    Fitting_1_2.append(np.mean(Slope_Fit_1_2))
    Fitting_1_2.append(np.mean(Offset_Fit_1_2))

    Mean_Frequency=np.mean(Freq)

    print(Print_Format+"For dataset %d, the fit parameters are: F = %f Hz/°C
* T °C + %f Hz"%(m,Fitting[0],Fitting[1]))
    print(Print_Format+"For dataset %d, the mean frequency is %f Hz" % (m,
Mean_Frequency))

    if Individual_Plotting == "y":
        Which_Plot = 'Partial_Drift_Plot'
        P_S = Plot_Settings_Modifier(Which_Plot)
        plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']), dpi=P_S['DPI'])
        plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
        plt.scatter(T_R_L_MM, Freq, cmap="gnuplot",
c=Time/prms['Seconds_To_Days'], marker="D", s=0.1)

        #plt.title('Temperature and frequency variations as a function of
time',y=1.05)
        plt.ylabel("Frequency [Hz]")
        plt.xlabel(r"$T_{Chamber}-20.5\ [\rm °C]$")
        Formating=ScalarFormatter(useOffset=True)
        Formating.set_useOffset(Mean_Frequency)
        plt.gca().yaxis.set_major_formatter(Formating)
        plt.gca().xaxis.set_major_formatter(StrMethodFormatter('{x:,.2f}'))
        Color_Map = plt.colorbar()
        Color_Map.set_label(Time_Label)

        Lower_Limit = np.min(T_R_L_MM) - np.min(T_R_L_MM) * 0.0025
        Upper_Limit = np.max(T_R_L_MM) + np.max(T_R_L_MM) * 0.0025
        Trend_Range = np.linspace(Lower_Limit, Upper_Limit, 1000)
        plt.plot(Trend_Range, np.polyval(Fitting, Trend_Range), 'g--',
linewidth=1)
        plt.show()

    return Fitting, Slope_Fit, Offset_Fit, Fitting_1_2, Slope_Fit_1_2,
Offset_Fit_1_2
#================================================================================
========================================#

#====================================== G - Individual Frequency Drift
Plotter ======================================#
def Plotting2(prms,df_Partial,Number_Of_Columns):
    t=df_Partial[0]
    Total_Time = df_Partial.iloc[-1][0]
    Timescale,Time_Label = Time_Scaler(Total_Time)
    FreqRel=df_Partial[Number_Of_Columns]
    Which_Plot = 'Partial_Drift_Plot'
    P_S = Plot_Settings_Modifier(Which_Plot)
```

```
    plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']), dpi=P_S['DPI'])
    plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
    plt.title("Frequency difference variations as a function of time")
    plt.ylabel("f - f0 [Hz]")
    plt.xlabel(Time_Label)
    plt.scatter(t/prms[Timescale],FreqRel , c='#ff940b', marker="D", s=0.25)
    plt.show()
#=========================================================================
=======================================#

#================================ H - Total Frequency Drift Plotter & Trend
Guess Saver ================================#
def
Plotting3(Temp_Comp,Time,Frequency,Frequency_1_2,Temperature,prms,f_initial,f
_initial_1_2,Time_Gap_Array,Filtering,Trend_Gen,Guess,Fit_Method):
    plt.rcParams.update({'font.size': 9})
    Which_Plot = 'Drift_Plot_With_Trend'
    P_S=Plot_Settings_Modifier(Which_Plot)
    fig1, F = plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']),
dpi=P_S['DPI'])
    plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
    F2 = F.twinx()
    Time_Collection = []
    Last_Time = Time[Number_Of_Folders-1]
    Total_Time = Last_Time[-1]
    Timescale,Time_Label = Time_Scaler(Total_Time)

    for m in range(Number_Of_Folders):
        if m == 0:
            Total_Time_Gap=0

        elif m != 0:
            Total_Time_Gap=Time_Gap_Array[m-1]+Final_Time

        Ti=Time[m]+Total_Time_Gap
        Final_Time = Ti[-1]
        Time_Collection=np.append(Time_Collection,Ti)

        Fr=Frequency[m]
        Fr_1_2=Frequency_1_2[m]

        Te=Temperature[m]


FrTemp=(Temp_Comp['Slope']*Te*(1+prms['Slope_Modifier_%']/100)+Temp_Comp['Off
set'])

FrTemp_1_2=(Temp_Comp['Slope12']*Te*(1+prms['Slope_Modifier_%']/100)+Temp_Com
p['Offset12'])


        if m == 0:
            FrRel = ((Fr-FrTemp)/f_initial)*1e6
            Initial_Zeroing_Value = FrRel[0]
            FrRel = FrRel - Initial_Zeroing_Value
```

```
            Artificial_Offset_Bottom=FrRel[-1]

            FrRel_1_2 = ((Fr_1_2-FrTemp_1_2)/f_initial_1_2)*1e6
            Initial_Zeroing_Value_1_2 = FrRel_1_2[0]
            FrRel_1_2 = FrRel_1_2 - Initial_Zeroing_Value_1_2
            Artificial_Offset_Bottom_1_2=FrRel_1_2[-1]

        elif m == 1:
            FrRel=((Fr-FrTemp)/f_initial)*1e6-Initial_Zeroing_Value
            Artificial_Offset_Top=FrRel[0]
            Artificial_Offset = np.abs(Artificial_Offset_Top-
Artificial_Offset_Bottom)
            FrRel = FrRel - Artificial_Offset

            FrRel_1_2=((Fr_1_2-FrTemp_1_2)/f_initial_1_2)*1e6-
Initial_Zeroing_Value_1_2
            Artificial_Offset_Top_1_2=FrRel_1_2[0]
            Artificial_Offset_1_2 = np.abs(Artificial_Offset_Top_1_2-
Artificial_Offset_Bottom_1_2)
            FrRel_1_2 = FrRel_1_2 - Artificial_Offset_1_2

        else:
            FrRel=((Fr-FrTemp)/f_initial)*1e6-Artificial_Offset-
Initial_Zeroing_Value
            FrRel_1_2=((Fr_1_2-FrTemp_1_2)/f_initial_1_2)*1e6-
Artificial_Offset_1_2-Initial_Zeroing_Value_1_2

        p1, = F.plot(Ti/Timescale, FrRel, 'green', linewidth=0.3,
label="Frequency [Hz]")
        F.tick_params(axis='y', colors=p1.get_color())

        p2, = F2.plot(Ti/Timescale, FrRel_1_2, 'cyan', linewidth=0.3,
label="Resonator Frequency Mode 1-2 [Hz]")
        F2.tick_params(axis='y', colors=p2.get_color())

        plt.draw()
        F.set_xlabel(Time_Label)
        F.set_ylabel("Drift [ppm]")
        F.yaxis.label.set_color(p1.get_color())

        F2.yaxis.label.set_color(p2.get_color())
        F2.set_ylabel("Drift [ppm]")
        F2.spines['right'].set_position(('outward', 0))
        m=+1

    if Filtering == 'y' and Trend_Gen == 1:
        for pos in ['right', 'top']:
            plt.gca().spines[pos].set_visible(False)
        if Fit_Method == 1:
            FrRel_Fit =
Guess['A1']*np.log(Guess['A2']*Time_Collection+Guess['t0a'])+Guess['B1']*np.l
og(Guess['B2']*Time_Collection+Guess['t0b'])
        elif Fit_Method == 2:
            FrRel_Fit =
Guess['A1']*np.log(Guess['A2']*Time_Collection+Guess['t0a'])+Guess['B1']*Time
_Collection+Guess['t0b']
        F.plot(Time_Collection /prms['Seconds_To_Days'], FrRel_Fit,
```

```
c='r',linewidth=0.75)

        custom_lines = [Line2D([0], [0], color='red', lw=1),Line2D([0], [0],
color='green', lw=1),Line2D([0], [0], color='cyan', lw=1)]
        F.legend(custom_lines, ['Fit','Mode 1-1 Drift','Mode 1-2
Drift'],loc='lower right', frameon=False,prop={'size': 6})
    plt.show()
#=============================================================================
=========================================#

#=================================== I - Trend Fitter & Guess Loader and
Saver ====================================#
def
Filtering_Fit(Temp_Comp,Time,Frequency,Temperature,prms,f_initial,Time_Gap_Ar
ray,Filtering, Fit_Method):
    if Filtering == 'y':

        Final_Rate = (input(Input_Format+'\nPlease input the final data
sampling rate, in Hz, that you wish to execute a curve fit for. Leave blank
for no filtering: '))
        if Final_Rate != '':
            Final_Rate=float(Final_Rate)

        Size_Before_Filter = 0
        Size_After_Filter = 0

        Time_Collection = []
        FrRel_Collection = []

        Last_Time = Time[Number_Of_Folders-1]
        Total_Time = Last_Time[-1]
        Timescale, Time_Label = Time_Scaler(Total_Time)

        plt.rcParams.update({'font.size': 9})
        Which_Plot = 'Test_Drift_Trend'
        P_S = Plot_Settings_Modifier(Which_Plot)
        fig1, F = plt.subplots(figsize=(P_S['x_dim'], P_S['y_dim']),
dpi=P_S['DPI'])
        plt.subplots_adjust(left=P_S['left'], right=P_S['right'],
bottom=P_S['bottom'], top=P_S['top'])
        plt.ylabel("Drift [ppm]")
        plt.xlabel(Time_Label)

        for m in range(Number_Of_Folders):
            Time_Fitting = Time[m]
            Frequency_Fitting = Frequency[m]
            Temperature_Fitting = Temperature[m]

            Length = len(Time_Fitting)
            Size_Before_Filter = Size_Before_Filter + Length
            Beginning_Time = Time_Fitting[0]
            End_Time = Time_Fitting[-1]
            Streaming_Rate = Length/(End_Time-Beginning_Time)

            if Final_Rate == '' or Final_Rate >= int(Streaming_Rate/2):
                Length = len(Time_Fitting)
                Size_After_Filter=Size_After_Filter+Length
```

```
                Interp_Time = Time_Fitting

            else:
                Filter_sos = sign.butter(N=4, Wn=Final_Rate,
fs=Streaming_Rate, output='sos')
                Interp_Time = np.linspace(Beginning_Time, End_Time,
int(Final_Rate * End_Time))

                Time_Fitting = sign.sosfiltfilt(Filter_sos, Time_Fitting)

                Frequency_Fitting = np.interp(Interp_Time, Time_Fitting,
sign.sosfiltfilt(Filter_sos, Frequency_Fitting))

                Temperature_Fitting = np.interp(Interp_Time, Time_Fitting,
sign.sosfiltfilt(Filter_sos, Temperature_Fitting))

                Length = len(Interp_Time)
                Size_After_Filter = Size_After_Filter + Length

            if m == 0:
                Total_Time_Gap = 0

            elif m != 0:
                Total_Time_Gap = Time_Gap_Array[m - 1] + Final_Time

            Ti = Interp_Time + Total_Time_Gap
            Final_Time = Ti[-1]
            Time_Collection = np.append(Time_Collection, Ti)
            Fr = Frequency_Fitting
            Te = Temperature_Fitting
            FrZero =
(Temp_Comp['Slope']*Te*(1+prms['Slope_Modifier_%']/100)+Temp_Comp['Offset'])
            if m == 0:
                FrRel = ((Fr-FrZero)/f_initial) * 1e6
                Initial_Zeroing_Value = FrRel[0]
                FrRel = FrRel - Initial_Zeroing_Value
                Artificial_Offset_Bottom = FrRel[-1]

            elif m == 1:
                FrRel = ((Fr - FrZero) / f_initial) * 1e6 -
Initial_Zeroing_Value
                Artificial_Offset_Top = FrRel[0]
                Artificial_Offset = np.abs(Artificial_Offset_Top -
Artificial_Offset_Bottom)
                FrRel = FrRel - Artificial_Offset

            else:
                FrRel = ((Fr-FrZero)/f_initial)*1e6-Artificial_Offset-
Initial_Zeroing_Value

            FrRel_Collection = np.append(FrRel_Collection, FrRel)

            plt.plot(Ti/Timescale, FrRel, c='green', linewidth=0.5)
            m = +1

        if Final_Rate == '' or Final_Rate >= int(Streaming_Rate/2):
            print(Print_Format+'\nData will not be filtered down further.')
```

```
            if Final_Rate != '' and Final_Rate >= int(Streaming_Rate/2):
                print(Print_Format+'The filtering rate must be higher than
half of the streaming rate: %d Hz. The data was thus not
filtered.'%(Streaming_Rate))
            Final_Rate = 'None'
        print(Print_Format+'\nThe size of a single data array before and
after filtering is respectively of %d and %d
points'%(Size_Before_Filter,Size_After_Filter))

        df_Fit_Guesses = pd.DataFrame()

        if Fit_Method == 1:
            df_Fit_Guesses = pd.read_csv('Fit_Guesses_Log_Log.txt',
sep=';',header=None)
            Guess = {'A1': df_Fit_Guesses.iloc[-1,1], 'A2':
df_Fit_Guesses.iloc[-1,3], 't0a': df_Fit_Guesses.iloc[-1,5], 'B1':
df_Fit_Guesses.iloc[-1,7], 'B2': df_Fit_Guesses.iloc[-1,9], 't0b':
df_Fit_Guesses.iloc[-1,11]}
            print('\nLoading the Fit Guesses: A1 = %f , A2 = %e , t0a = %f ,
B1 = %f , B2 = %e , t0b = %f'%
(Guess['A1'],Guess['A2'],Guess['t0a'],Guess['B1'],Guess['B2'],Guess['t0b']))
            popt, pcov = curve_fit(Log_Log_Fit, Time_Collection,
FrRel_Collection,p0=(Guess['A1'], Guess['A2'], Guess['t0a'], Guess['B1'],
Guess['B2'], Guess['t0b']),bounds=((-np.inf, 0, 0, 0, 0, 0), (0, np.inf,
np.inf, np.inf, np.inf, np.inf)),method='dogbox')
            Coef = {'A1': popt[0], 'A2': popt[1], 't0a': popt[2], 'B1':
popt[3], 'B2': popt[4], 't0b': popt[5]}
            print(Print_Format+"\nThe first log equation is:
FrRel_A=%.2f*ln(%.2e*t+%.2f) and the second is FrRel_B=%.2f*ln(%.2e*t+%.2f)"
% (Coef['A1'],Coef['A2'],Coef['t0a'],Coef['B1'],Coef['B2'],Coef['t0b']))
            FrRel_Fit =
Coef['A1']*np.log(Coef['A2']*Time_Collection+Coef['t0a'])+Coef['B1']*np.log(C
oef['B2']*Time_Collection+Coef['t0b'])

        if Fit_Method == 2:
            df_Fit_Guesses = pd.read_csv('Fit_Guesses_Log_Lin.txt',
sep=';',header=None)
            Guess = {'A1': df_Fit_Guesses.iloc[-1,1], 'A2':
df_Fit_Guesses.iloc[-1,3], 't0a': df_Fit_Guesses.iloc[-1,5], 'B1':
df_Fit_Guesses.iloc[-1,7], 't0b': df_Fit_Guesses.iloc[-1,9]}
            print('\nLoading the Fit Guesses: A1 = %f , A2 = %e , t0a = %f ,
B1 = %e, t0b = %f'%
(Guess['A1'],Guess['A2'],Guess['t0a'],Guess['B1'],Guess['t0b']))
            popt, pcov = curve_fit(Log_Lin_Fit, Time_Collection,
FrRel_Collection,p0=(Guess['A1'], Guess['A2'], Guess['t0a'], Guess['B1'],
Guess['t0b']),bounds=((-np.inf, 0, 0, 0, -np.inf), (0, np.inf, np.inf,
np.inf, 0)),method='trf')
            Coef = {'A1': popt[0], 'A2': popt[1], 't0a': popt[2], 'B1':
popt[3],'t0b': popt[4]}
            print(Print_Format+"\nThe log equation is
FrRel_A=%.2f*ln(%.2e*t+%.2f) and the linear equation is FrRel_B=%.2e*+%.2f" %
(Coef['A1'],Coef['A2'],Coef['t0a'],Coef['B1'],Coef['t0b']))
            FrRel_Fit =
Coef['A1']*np.log(Coef['A2']*Time_Collection+Coef['t0a'])+Coef['B1']*Time_Col
lection+Coef['t0b']

        plt.plot(Time_Collection/Timescale, FrRel_Fit, c='r',linewidth=1)
```

```
        plt.yticks([-250,-125,0],['-250','-125', '0'])
        plt.show()

        Condition =1

        while Condition == 1:
            Satisfaction=input(Input_Format+'\nAre you satisfied with the
curve fit? Answer "y" to continue with curent fit and save its guesses in
"Fit_Guesses.txt" or "n" to restart the fit with a new filtering value: ')
            if Satisfaction.casefold() == 'y':
                Fit_Satisfaction = 1
                Fit_Date=datetime.now()
                if Fit_Method == 1:
                    Fit_Guesses_File_Creation =
open("Fit_Guesses_Log_Log.txt", "a+")
                    Fit_Guesses_File_Creation.write("\nA1 = ; %f ; A2 = ; %e
; t0a = ; %f ; B1 = ; %f ; B2 = ; %e ; t0b = ; %f ; Additional Filtering
Final Rate [Hz] ; %s ; %s" %(popt[0], popt[1], popt[2], popt[3], popt[4],
popt[5],Final_Rate,Fit_Date))
                    Fit_Guesses_File_Creation.close()
                elif Fit_Method ==2:
                    Fit_Guesses_File_Creation =
open("Fit_Guesses_Log_Lin.txt", "a+")
                    Fit_Guesses_File_Creation.write("\nA1 = ; %f ; A2 = ; %e
; t0a = ; %f ; B1 = ; %e ; t0b = ; %f ; Additional Filtering Final Rate [Hz]
; %s ; %s" %(popt[0], popt[1], popt[2], popt[3],
popt[4],Final_Rate,Fit_Date))
                    Fit_Guesses_File_Creation.close()
                Condition = 0
            elif Satisfaction.casefold() == 'n':
                Fit_Satisfaction = 0
                Condition = 0
            else:
                print(Warning_Format+"\nInvalid answer. Please restart.")
                Condition = 1

    else:
        Coef = {'A1': None, 'A2': None, 't0a': None, 'B1': None, 'B2': None,
't0b': None}
        Fit_Satisfaction = 1

    return Coef,Fit_Satisfaction
#================================================================================
=========================================#

#============================================ 1 - Setting Text Formating
==========================================#
Input_Format=Fore.GREEN+Style.BRIGHT
Warning_Format=Fore.RED+Style.BRIGHT
Print_Format=Fore.RESET+Style.BRIGHT
#================================================================================
=========================================#

#============================================ 2 - Material Properties
==========================================#
ESiN = 250000
#Silicon nitride modulus of elasticity [MPa]
```

```
VSiN = 0.23
#Silicon nitride poisson's ratio
CTE_Si = 2.6e-6
#Silicon CTE
CTE_SiN = 2.3e-6
#Silicon nitride CTE
E_Prime_SiN = ESiN/(1-VSiN)
#Modified modulus of elasticity [MPa]
Sigma_Zero = 100
#Silicon nitride initial stress [MPa]
dct_prop = {'CTE_Si':CTE_Si, 'CTE_SiN':CTE_SiN, 'E_Prime_SiN':E_Prime_SiN,
'Sigma_Zero':Sigma_Zero}
#================================================================================
========================================#

#========================================= 3 - Data Processing Parameters
========================================#
Drive_Label = 'A'
Lab_Temperature = 20.5
Averaging_Interval = 6
Seconds_To_Days=60*60*24
Averaging_Interval_Seconds=Averaging_Interval*60*60
Slope_Modifier_Percentage = 8
print(Print_Format+"Averaging at a rate of %d hours\n"%(Averaging_Interval))
dct_param =
{'Lab_Temperature':Lab_Temperature,'Averaging_Interval_Seconds':Averaging_Int
erval_Seconds,'Slope_Modifier_%':Slope_Modifier_Percentage,'Seconds_To_Days':
Seconds_To_Days}
#================================================================================
========================================#

#========================================== 4 - Choice of Data to Load
==========================================#
Condition = 1
while Condition == 1:
    Path_Choice = int(input(Input_Format+'Which datalog do you wish to
analyse:\n          1) 1 Mode With Vacuum Leak\n          2) 2 Modes Good
Aging Data\n          3) 2 Modes Outside Fridge\nAnswer with either 1, 2 or
3: '))
    if Path_Choice == 1:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\1 Mode With Vacuum Leak'
        Condition = 0
    elif Path_Choice == 2:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\2 Modes Good Aging Data'
        Condition = 0
    elif Path_Choice == 3:
        Mainpath = r':\OneDrive - University of Ottawa\01 UOMEMS Student
Project Folder - SHARED\Michel Stephan\12 - Stability Measurement\Cell - 6 mm
- Low Stress\Lock-In With Multimeter\2 Modes Outside Fridge'
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.\n")
```

```
Condition = 1
while Condition == 1:
    File_To_Use = int(input(Input_Format+'\nWould you liked to use:\n
1) Merged Filtered Data\n          2) Merged Filtered Data Time
Compensated\nAnswer with either 1 or 2: '))
    if File_To_Use == 1:
        File_Termination = "\Datasets\Merged_Filtered_Data.txt"
        Condition = 0
    elif File_To_Use == 2:
        File_Termination =
"\Datasets\Merged_Filtered_Data_Time_Compensated.txt"
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.")

Condition = 1
while Condition == 1:
    Individual_Plotting = input(Input_Format+"\nWould you like to generate
plots for every dataset? [y/n] ")
    Individual_Plotting = Individual_Plotting.casefold()
    if Individual_Plotting == 'y':
        Condition = 0
    elif Individual_Plotting == 'n':
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.")

Condition = 1
while Condition == 1:
    Filtering = input(Input_Format+"\nWould you like to generate a trend for
the frequency drift plot? [y/n] ")
    Filtering = Filtering.casefold()
    if Filtering == 'y':
        Condition = 0
    elif Filtering == 'n':
        Condition = 0
    else:
        print(Warning_Format+"\nInvalid answer. Please restart.")

Condition = 1
while Condition == 1:
    Fit_Method = int(input(Input_Format + 'Which fitting function do you wish
to use:\n          1) Log-Log Model\n          2) Log-Linear Model\nAnswer
with either 1 or 2: '))
    if Fit_Method == 1:
        Condition = 0
    elif Fit_Method == 2:
        Condition = 0
    else:
        print(Warning_Format + "\nInvalid answer. Please restart.\n")
#=========================================================================
=======================================#

#====================== 5 - Loading Data, Large Dataframe Creation & Thermal
Compensation Trend=======================#
Filepath=os.listdir(Drive_Label+Mainpath)
Number_Of_Folders = len(Filepath)
```

```
j = 0

df_Total = pd.DataFrame()
Linear_Fit_Parameters = []

Temp_Rel_Lab_MM=[]

Slope_Fit_Storage = []
Offset_Fit_Storage = []
Slope_Fit_Storage_1_2 = []
Offset_Fit_Storage_1_2 = []

Time = []
Frequency = []
Frequency_1_2=[]
Temperature = []

Date_Array_Start = []
Date_Array_Duration = []
Time_Gap_Array = []

m=0

while j < Number_Of_Folders:
    print(Print_Format+"\nWorking on storing dataset with index %d in a
dataframe" % (j))
    i = 0
    index = str(i)
    df_Partial = pd.DataFrame()
    df_Partial = pd.read_csv(Drive_Label + Mainpath +'\\'+
Filepath[j]+'\Datafiles'+File_Termination, sep=';',skiprows=1, header=None)

    Date_File =
open(Drive_Label+Mainpath+'\\'+Filepath[j]+'\Datafiles'+File_Termination,
'r')
    Line_Text = Date_File.readline()
    Found_Date = list(datefinder.find_dates(Line_Text))
    if len(Found_Date) > 1:
        Found_Date = str(Found_Date[2])
    else:
        Found_Date = str(Found_Date[1])
    Stripped_Date=datetime.strptime(Found_Date,"%Y-%m-%d %H:%M:%S.%f")
    Date_Array_Start.append(Stripped_Date)
    Date_Array_Duration.append(df_Partial.iloc[-1][0])

    if j != 0:
        Time_Gap_Array.append((Date_Array_Start[j]-Date_Array_Start[j-
1]).total_seconds()-Date_Array_Duration[j-1])

    if j == 0:
        f_initial=df_Partial.iloc[0][1]
        f_initial_1_2=df_Partial.iloc[0][4]

    Number_Of_Columns = len(df_Partial.columns)

    Total_Time = df_Partial.iloc[-1][0]
```

```
    print(Print_Format+"The total duration of the analysed data is of %f
days\n" % (float(Total_Time)/dct_param['Seconds_To_Days']))

    T_MM_Relative_Lab=df_Partial[2]-(273.15)-(Lab_Temperature)

    Fitting, Slope_Fit, Offset_Fit, Fitting_1_2, Slope_Fit_1_2,
Offset_Fit_1_2=Plotting(m,dct_param,df_Partial,T_MM_Relative_Lab,Individual_P
lotting)

    df_Partial[Number_Of_Columns]=df_Partial[1]-
(Fitting[0]*T_MM_Relative_Lab+Fitting[1])
    df_Partial[Number_Of_Columns+1] = df_Partial[4]-(Fitting_1_2[0] *
T_MM_Relative_Lab + Fitting_1_2[1])

    if Individual_Plotting == "y":
        Plotting2(dct_param,df_Partial,Number_Of_Columns)

    m+=1

    Slope_Fit_Storage=np.concatenate([Slope_Fit_Storage,Slope_Fit])
    Offset_Fit_Storage=np.concatenate([Offset_Fit_Storage,Offset_Fit])

Slope_Fit_Storage_1_2=np.concatenate([Slope_Fit_Storage_1_2,Slope_Fit_1_2])

Offset_Fit_Storage_1_2=np.concatenate([Offset_Fit_Storage_1_2,Offset_Fit_1_2]
)

    Time.append(df_Partial[0].to_numpy())
    Frequency.append(df_Partial[1].to_numpy())
    Frequency_1_2.append(df_Partial[4].to_numpy())
    Temperature.append((df_Partial[2]-273.15-Lab_Temperature).to_numpy())

    j+=1

dct_temp_comp =
{'Slope':np.mean(Slope_Fit_Storage),'Offset':np.mean(Offset_Fit_Storage),'Slo
pe12':np.mean(Slope_Fit_Storage_1_2),'Offset12':np.mean(Offset_Fit_Storage_1_
2)}

print(Print_Format+"\nFor the merged datasets, the linear equation obtained
for mode 1-1 is: F = %f Hz/°C * T °C + %f Hz"
%(dct_temp_comp['Slope'],dct_temp_comp['Offset']))
print(Print_Format+"For the merged datasets, the linear equation obtained for
mode 1-2 is: F = %f Hz/°C * T °C + %f Hz"
%(dct_temp_comp['Slope12'],dct_temp_comp['Offset12']))
#=========================================================================
=======================================#

#========================================== 6 - Drift Ploting and Fitting
=========================================#
dct_guesses = {'A1':0,'A2':0,'t0a':0,'B1':0,'B2':0,'t0b':0}

Trend_Gen = 0
Plotting3(dct_temp_comp,Time,Frequency,Frequency_1_2,Temperature,dct_param,f_
initial,f_initial_1_2,Time_Gap_Array,Filtering,Trend_Gen,dct_guesses,Fit_Meth
od)
```

```
Trend_Gen = 1

Fit_Satisfaction = 0
while Fit_Satisfaction == 0:

dct_guesses,Fit_Satisfaction=Filtering_Fit(dct_temp_comp,Time,Frequency,Tempe
rature,dct_param,f_initial,Time_Gap_Array,Filtering,Fit_Method)

Plotting3(dct_temp_comp,Time,Frequency,Frequency_1_2,Temperature,dct_param,f_
initial,f_initial_1_2,Time_Gap_Array,Filtering,Trend_Gen,dct_guesses,Fit_Meth
od)
#=========================================================================
========================================#
```

Rate and Aging Source Calculations.py

```
#============================================== A - Importing Libraries
==============================================#
print("\nRead 'Frequency Drift Plotter README.txt' for any information about
this script.")
print('\nLoading Libraries...')
import numpy as np
print('\nLibraries Loaded!\n')
#============================================================================
=========================================#

#=================================== B - Stress Frequency Drift Conversion
=========================================#
def Stress_To_Frequency_Drift_ppm(Sigma,Sigma_Initial):
    Frequency_Drift_ppm = 1/2*Sigma/Sigma_Initial*1e6
    return Frequency_Drift_ppm
#============================================================================
=========================================#

#=================================== C - Mass Frequency Drift Conversion
=========================================#
def Mass_To_Frequency_Drift_ppm(Mass_Differential,Mass):
    Frequency_Drift_ppm = -1/2*Mass_Differential/Mass*1e6
    return Frequency_Drift_ppm
#============================================================================
=========================================#

#========================================= 1 - Adsorption & Desorption
Rates=========================================#
NA = 6.022e23
#1/mol
M_H = 1.008
#g/mol
M_O = 15.999
#g/mol
M_H2O = M_H*2+M_O
#g/mol
m_H20 = M_H2O/NA
#g
m_H20_kg=m_H20/1000
#kg
P_Torr_Vacuum = 1.65e-7
#Torr
P_Pa_Vacuum = 10e-5
#Pa <-> kg/(m*s^2) <-> J/m^3
P_Pa_Atmorspheric = 1250
#Pa <-> kg/(m*s^2) <-> J/m^3
Eb_kcal_mol = 5.5
#kcal/mol
Eb_joule = Eb_kcal_mol/NA*4184
#J/molecule
Eb_joule = 1.1e-19
#Choose Eb_kcal or Eb_joule
Kb = 1.38e-23
#J/K WHERE J=(kg*m^2)/s^2
```

```
T = 295
#K
vd = 1e13
#1/s
s = 0.1

ra = s*(2/5)*P_Pa_Vacuum/np.sqrt(m_H20_kg*Kb*T)
ra_atmospheric= s*(2/5)*P_Pa_Atmorspheric/np.sqrt(m_H20_kg*Kb*T)

Site_Area = 1/1e19
ra_site = ra*Site_Area
ra_atmospheric_site=ra_atmospheric*Site_Area

rd = vd*np.exp(-(Eb_joule/(Kb*T)))

f = ra_site/(ra_site+rd)
frac_occ_rel = ra_atmospheric_site/(ra_atmospheric_site+rd)

print('\nAdsorption rate at ambient pressure is: ra = %e s^-
1'%(ra_atmospheric_site))
print('Adsorption rate at vacuum pressure is: ra = %e s^-1'%(ra_site))
print('Desorption rate rd = %e s^-1\n'%(rd))
print(f'The ambient active site occupancy is of f_atm =
{(frac_occ_rel*100):.2e} %')
print(f'The vacuum active site occupancy is of f = {(f*100):.2e} %\n')
#=============================================================================
=========================================#

#========================================== 2 - Mass Adsorption Induced
Drift=========================================#
#Membrane Dimension & Property#
L1=6 #mm
L2=6 #mm
t=90e-9 #nm
Sigma_Initial = 100e6


#Molecule Properties#
NA = 6.02214076e+23

M_H = 1.008
M_O = 15.999
M_Si = 28.085
M_N = 14.007

#H2O Properties#
Rho_H2O=997 #kg/m3
M_H2O = M_H*2+M_O
m_H2O = M_H2O/NA
print("The molecular mass of H2O is: %.3f g/mol"%(M_H2O))
print("The mass of a single molecule of H2O is: %.3e g"%(m_H2O))

#Si3N4 Properties#
Rho_SiN=3170 #kg/m3
Membrane_Mass = L1/1000*L2/1000*t*Rho_SiN*1000
Membrane_Area = (6/1000)**2
M_SiN = M_Si*3+M_N*4
```

```
m_SiN = M_SiN/NA
Rho_SiN_Sites_Volumetric = Rho_SiN/(M_SiN/1000)*NA
Rho_SiN_Sites_Area = Rho_SiN_Sites_Volumetric**(2/3)
N_SiN_Sites_Membrane = Rho_SiN_Sites_Area*Membrane_Area

print('\nThe volumetric and surface site density if respectively of %.3e
sites/m3 and %.3e sites/m2'%(Rho_SiN_Sites_Volumetric,Rho_SiN_Sites_Area))

print("\nThe molecular mass of SiN is: %.3f g/mol"%(M_SiN))
print("The mass of a single molecule of SiN is: %.3e g"%(m_SiN))

print("The maximal number of SiN molecule that can fit in a monolayer of area
%dx%d mm^2 is of: %d molecules"%(L1,L2,N_SiN_Sites_Membrane))
print("The total mass of the SiN membrane of dimension %dmmx%dmm%dnm is of:
%.3e g"%(L1,L2,t,Membrane_Mass))

#Total Mass#
T_m_SiN = m_SiN*N_SiN_Sites_Membrane
T_m_SiN_H2O = (m_SiN+m_H2O)*N_SiN_Sites_Membrane
Adsorption_Mass_Differential = T_m_SiN_H2O-T_m_SiN

print("\nThe total mass of molecules of SiN that fit in a monolayer of area
%dx%d mm^2 is of: %.3e g"%(L1,L2,T_m_SiN))
print("The total mass of molecules of SiN with adsorbed H20 that fit in a
monolayer of area %dx%d mm^2 is of: %.3e g"%(L1,L2,T_m_SiN_H2O))
print("The total mass of molecules of H20 adsorbed for a monolayer of area
%dx%d mm^2 is of: %.3e g"%(L1,L2,T_m_SiN_H2O-T_m_SiN))

FrRel_Mass_Drift =
Mass_To_Frequency_Drift_ppm(Adsorption_Mass_Differential,Membrane_Mass)
FrRel_Mass_Drift_Frac_Occ = FrRel_Mass_Drift*f
print("\nThe frequency shift induced by the adsorption of H2O for a SiN
monolayer of area %dx%d mm^2 is of: %.3f ppm"%(L1,L2,2*FrRel_Mass_Drift))
print("The frequency shift induced for both of our membrane's side and
considering the fractional site occupancy is: %.3f
ppm"%(2*FrRel_Mass_Drift_Frac_Occ))
#=========================================================================
=======================================#


#=================================== 3 - Water Tension Stress Induced
Drift=======================================#
delta_s_water = 73e-3
Sigma_Surface_Tension_Water = delta_s_water/t
FrRel_Water_Tension =
Stress_To_Frequency_Drift_ppm(Sigma_Surface_Tension_Water,Sigma_Initial)
FrRel_Water_Tension_Frac_Occ = FrRel_Water_Tension*f
print("\nThe frequency shift induced by the surface tensions stress from H2O
adsorption for a SiN monolayer of area %dx%d mm^2 is of: %.3f
ppm"%(L1,L2,2*FrRel_Water_Tension))
print("The frequency shift induced by the surface tensions stress from H2O
adsorption considering the fractional site occupancy is: %.3f
ppm"%(2*FrRel_Water_Tension_Frac_Occ))
#=========================================================================
=======================================#
```

```
#=================================== 4 - Chemically Tension Stress Induced
Drift=====================================#
delta_s_chemical = 1.1
Sigma_Surface_Tension_Chemical = delta_s_chemical/t
FrRel_Chemical_Tension =
Stress_To_Frequency_Drift_ppm(Sigma_Surface_Tension_Chemical,Sigma_Initial)
FrRel_Chemical_Tension_Frac_Occ = FrRel_Chemical_Tension*f
print("\nThe frequency shift induced by the chemically induced surface
tensions stress for a SiN monolayer of area %dx%d mm^2 is of: %.3f
ppm"%(L1,L2,2*FrRel_Chemical_Tension))
print("The frequency shift induced by the chemically induced surface tensions
stress considering the fractional site occupancy is: %.3f
ppm"%(2*FrRel_Chemical_Tension_Frac_Occ))
#=============================================================================
=========================================#
```

B. Python Scripts README

Continuous Recording README.txt

```
The goal of this script is to record the raw data for the long-term stability
study of our SiN membranes.
This script has been created by Michel Stephan, using Zurich's example script
"Continuous Recording.txt".
This script, following the proper setup of the LabOne interface, has also
specifically been adapted to
stream and record the following data from a Zurich Lock-In amplifier and a
Keithley DMM6500 Multimeter.

    Zurich MFLI Lock-In Amplifier:  Demodulator 1 (Mode 1-1 Frequency, Mode
1-1 Voltage Magnitude, Demodulator 1 Time)
                                    Demodulator 2 (Mode 1-2 Frequency, Mode
1-2 Voltage Magnitude, Demodulator 2 Time)
                                    Demodulator 3 (Fringe Voltage Magnitude,
Demodulator 3 Time)

    Keithley DMM6500 Multimeter:   Channel 2+7 (Vacuum Cell Thermocouple's 4-
Wire Resistance)
                                    Channel 3 (Lab RTD Resistance)

All of the streamed information is stored into arrays that will be saved
periodically into specific text files with a
strict formatting. There are timing settings to choose the frequency at which
the stored data is saved into a text
files, to choose how much time the continuous recorded should go on as well
as to choose the frequency of a status
messages. See section 5 - Timing & Interval Selector of script to quickly
modify these settings.

NOTE: the format and name of the saved textfile is specific and proprietary
to the other scripts that manipulate this
raw data. If the structure, name or amount of data saved is changed, the
other scripts will require some modifications
as well.

NOTE: The continuous recording part of the script that relates to the Lock-In
Amplifier is based on the example
provided by Zurich on their website. More information about the specific
Zurich Python commands and settings can be
found in these provided example scripts or in the "Continuous Recording.txt"
file.

IMPORTANT NOTE: For the Keithley DMM6500 data streaming to work correctly,
the user must ensure that the "functions.lua" file is present in
       the same folder as the script and that the used Python IDE (Pycharm,
Spyder, etc) LUA plugins as enabled.

SCRIPT OVERVIEW (Following section labels):
        A - Loads the required libraries
    B - Function used to stream data from the lock-in amplifier and the
```

```
multimeter. The obtained streamed data is directly stored in arrays
        Having the multimeter measurements in that configuration inside the
function gives it a streaming rate of 30 Hz for the thermistor data
    1 - Section of the script that connects to the devices.
    2 - Section used to set up the different signals of the lock-in amplifier
that will be recorded by entering their proprietary path
    3 - Section used to set up the different signals and channels of the
multimeter and set the proper range of the recorded resistance data
    4 - Section used to set up the different arrays and dictionaries that
will be used to store the streamed data
    5 - Section used to set up the different timing setting such as the
maximum logging time, interval of saving and update interval
    6 - Section used to set up the saving path as well as the headers of the
saved textfiles
    7 - Section used to specify the settings of the lock-in streaming
parameter. For more details, see Zurich's script "Continuous Recording.txt"
    8 - Section used to subscribe to the lock-in amplifier signals, which is
required when streaming, selected in Section 2.
    9 - Section used to stream the RTD data from the multimeter, gather all
the streamed data stored into array and save them into textfiles with a
specific format
    10 - Section used save the final bits of data that might have been
streamed after the burst requested to Section and safely close the
connections to the instruments

Structure of saved data:
    The saved data follows a structure that is determined throughout the
script and that is based around the dictionaries set up in Section 4.
    To change the main location to save the data, simply modify the
"Save_Path" variable in the Section 6 of the script.

    To change the type of data streamed its structure data, one has to
modify:
       The lock-in signals we wish to record by following the proprietary
path nomenclature in Section 2
       The multimeter signals we wish to record in Section 3, by using SCPI
coding language and the specific commands found in the DMM6500 User's Manual
       The structure of the dictionaries which will hold data to the desired
one, in Section 4
       The structure and format of the header for the textfile to the desired
one, in Section 6
       The location of the storage for data streamed in Section B, Section 9
and Section 10 to match the new dictionary
       The content of the arrays being saved in Section 9 and Section 10 to
contain the new streamed information properly

    Current Lock-in Textfile Structure:
       DM1 Frequency [Hz] ; DM1 F Time [s] ; DM1 Voltage [V] ; DM2 Frequency
[Hz] ; DM2 Voltage [V] ; DM2 V Time [s] ; DM3 Voltage [V] ; DM3 V Time [s] ;
<YEAR/MONTH/DAY HOUR:MINUTE:SECOND.DECIMAL when data is saved>

    Current Multimeter Thermistor Textfile Structure:
       Thermistor 4-W Resistance [Ohm] ; Resistance Time  [s] ;
<YEAR/MONTH/DAY HOUR:MINUTE:SECOND.DECIMAL when data is saved>

    Current Multimeter RTD Textfile Structure:
       RTD 2-W Resistance [Ohm] ; Resistance Time  [s] ; <YEAR/MONTH/DAY
```

```
HOUR:MINUTE:SECOND.DECIMAL when data is saved>

Timing settings:
    This script allows the used to have different timing settings for three
options:
       1- Total duration of the logging period
       2- Sampling rate of the lock-in amplifier data (Currently once per 5
minutes)
       3- Duration of the burst of lock-in data stream (Recommended to be
left as 1)
       4- Time interval before the streamed and stored data is saved in
textfiles
       5- Sampling rate of the multimeter RTD data
       6- Time interval to print an update status reporting the current time
and script recording status

    To modify these settings, one has to respectively change the values of
the variables in Section 5, to the desired time in seconds.
    The variables that match the timing settings above, are respectively:
       1- "total_duration"
       2- "module_sampling_rate"
       3- "burst_duration"
       4- "Recording_time_limit"
       5- "Recording_time_limit_RTD"
       6- "Checkpoint_Interval"
```

Merging and Filtering README.txt

```
The goal of this script is to allow the user to merge datafiles from a Zurich
MFLI Lock-In Amplifier and a Keithley
DMM6500 Multimeter into datasets. This script is complementary to the
"Continuous Recording" script and has been created by Michel Stephan.

This script also has a GUI that allows the user to quickly plot different
data from the datafiles and apply signal processing.

Definitions:
    Datafile: Single text file with data streamed from the LockIn/Multimeter.
Produced by 'Continuous Recording' script
    Dataset: Collection of all single text files for the continuous session
of streaming

This script has been developed to use the following format of file path and
names to function wit no error:
        Folder Structure: '[Drive Label]:\[Custom Path]\Datafiles\Datasets'
        Proprietary Path: '[Drive Label]:\[Custom Path]\Datafiles'

This script also gives the users instruction and options. Here is the
following text legend:
        Green Text: User Input
        White Text: Script Status Information
        Red Text: Warning Invalid Input

This script has two execution options:
        1) Generate Merged Filtered Dataset
        2) Load Pre-existing Merged Filtered Dataset

A user should always begin by option 1) to generate a dataset from his
datafiles. He can then use this dataset to
compute the Time-Lag Compensation or he can, in a later instance, load this
dataset with option 2) to execute the time
compensation.

In any case, to be able to use the Quick Plot GUI, the user will have to
select option 1.

SCRIPT OVERVIEW (Following section labels in script comments):
        A - Loads the required libraries
    B - Function used to format the time label and timescale used in plotting
    C - Function used to update the Quick Plot GUI and ensure plots can be
redrawn
    D - Function used to save the parameters for the signal processing
settings used in the Quick Plot GUI so it can be truly applied to the data
later in the script
    E - Function used to close and safely destroy the Quick Plot GUI
    F - Function used to create the Quick Plot GUI and make all of its
functionalities executable
    G - Function used to find the peaks in the loaded data through the
modifiable and designed algorithm
        1 - Setting the text formatting
        2 - Loads path with required files
        3 - Ask the user for his execution option
```

```
    If Option 1 is chosen:
        4.1 - Loads datafiles from Proprietary Path and merges them in a
Pandas Dataframe
        4.2 - Converts the resistance of the Multimeter's Thermistor and RTD
(if available) into a temperature value using the appropriate equations
                Thermistor information:
https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&Doc
Id=Data+Sheet%7FTESS-ANDO-DS0089%7FA%7Fpdf%7FEnglish%7FENG_DS_TESS-ANDO-
DS0089_A.pdf%7F11029674-00
                100 Ohm RTD information:
https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&Do
cNm=PTF-
FAMILY&DocType=DS&DocLang=English&fbclid=IwAR1i_xfS_E3KMcrFrHugaxvSnl3OjmrtGX
CYZF2JfvurHdmInMagzuRak98
        4.3 - Initializes timestamp depending on size of amount of different
signals recorded from the Lock-In
        4.4 - Generates the Quick Plot GUI
        4.5 - Trims and applies a rolling median if it was selected in the
GUI
        4.6 - Interpolates a global timestamp and filters the data to a final
rate (Hz) (Line 598 in script)
                Interpolation Format: (Original timestamp of signal, Rolling
Median timestamp of the signal being filtered, Signal to filter)
        4.7 - Section that saves the merged dataset if the user wishes to,
and then proceed to 5.1
    If Option 2 is chosen:
        5.1 - Section that uses the merged dataset, does a lag compensation
and saves the compensated dataset if the user wishes to

Quick Plot GUI:
        The Quick Plot GUI is a tool created to quickly plot and visualize
the recording signals using different timestamps
        while also giving the option to trim the beginning or end of the
dataset and even apply a rolling median.

        Signal Plotting:
           To plot desired signal, simply tick the boxes from the 'Plotted
Curves' section, select the timestamp from the
           'Timestamp Type' section (defaulted to 'Seconds [s]' if nothing is
chosen) and then click the 'Plot' button. To plot
           with new settings, depending on the IDE used, close the active
plot, select new settings as explained previously and
           hit the 'Plot' button.

        Trimming Preview:
           The Quick Plot GUI is designed to give you the option to preview
some trimming and rolling median as well as to
           really apply it to the dataset. Simply input your desired trimming
start and end based on the plot you generated and
           then hit the 'Plot' button.
           WARNING, the trimming numbers must be in the same unit as the ones
selected in the 'Timestamp Type' section.

        Rolling Median Preview:
           Simply slide the rolling median marker to the value desired and
then hit the 'Plot' button to see the preview.
```

```
        To save the trimming or median settings and really apply it to the
dataset, input the desired parameters and then
        hit the 'Save' button.

        To exit the GUI and proceed to the next section of the script, hit
the 'Exit' button.

Peak Finder Algorithm:
        The peak finder algorithm finds peaks in the multimeter and lock-in
signals. The peaks from both signal are then
        compared one to another to ensure that they are matching peaks. If
the peaks are not matching, they will be removed
        from the signal with the most peaks found, until an even amount of
peaks that are matching are left between both
        signals.

        The peaks are found using these following parameters:
            Minimum: Minimum threshold to be considered as a peak.
            Prominence: Minimum distance required between the closest local
minima and peak.
            Distance: Distance on the x-axis required between two peaks.

        If the peak finding algorithm gives unusual results, you have to
modify its peak finding parameters found in Function G.
            Peak_Delta: Controls the time margin in seconds between a peak
from the Multimeter and Lock-In to be
                                            considered as matching peaks.
            Peak_Distance: Minimum distance required between two peaks.
Highly recommended to be the same as
                  Peak_Delta to ensure that the peaks from each signal might
not be matched with more
                  than one peak.
                                            In mathematical terms: 0<LI_Time_Peak-
MM_Time_Peak<Peak_Delta to be a matching peak.
            LI_Prominence: The minimum prominence of a peak in the Lock-In
signal. If unsure, select inspect the
                                           minimum distance between a local minima
and a peak you wish to use as a reference and
                                           use the value.
            MM_Prominence: The minimum prominence of a peak in the Multimeter
signal. If unsure, select inspect
                                           the minimum distance between a local
minima and a peak you wish to use as a reference
                                           and use the value.

Predefined Format of Files:
        This script was developed with set types of files formats. Each
datafile has a header containing the name of each column,
    and the exact time when the file was produced. The format of these files
is set in the 'Continuous Recording' script along
    with the options selected in the Zurich MFLI's Lab One web user
interface.

    LockIn_'Index' (Version with 4 column):
       Column 0: Demodulator 1 Frequency (Hz)
       Column 1: Demodulator 1 Frequency Timestamp (s)
       Column 2: Demodulator 1 Voltage (V)
```

```
       Column 3: Demodulator 1 Voltage Time (s)

    LockIn_'Index' (Version with 8 column):
       Column 0: Demodulator 1 Frequency (Hz)
       Column 1: Demodulator 1 Frequency Timestamp (s)
       Column 2: Demodulator 1 Voltage (V)
       Column 3: Demodulator 2 Frequency (Hz)
       Column 4: Demodulator 2 Frequency Timestamp (s)
       Column 5: Demodulator 2 Voltage (V)
       Column 6: Demodulator 3 Voltage (V)
       Column 7: Demodulator 3 Timestamp (V)

    LockIn_'Index' (Version with 10 column):
       Column 0: Demodulator 1 Frequency (Hz)
       Column 1: Demodulator 1 Frequency Timestamp (s)
       Column 2: Demodulator 1 Voltage (V)
       Column 3: Demodulator 1 Voltage Timestamp (s)
       Column 4: Demodulator 2 Frequency (Hz)
       Column 5: Demodulator 2 Frequency Timestamp (s)
       Column 6: Demodulator 2 Voltage (V)
       Column 7: Demodulator 2 Voltage Timestamp (s)
       Column 8: Demodulator 3 Voltage (V)
       Column 9: Demodulator 3 Timestamp (V)

    Multimeter_'Index':
       Column 0: Thermistor Resistance (Ohm)
       Column 1: Timestamp (s)

    Multimeter_RTD_'Index':
       Column 0: RTD Resistance (Ohm)
       Column 1: Timestamp (s)

Saved Data:
    This script will thus produce, depending on the chosen option, merged and
filtered datasets containing the following information:
    For the Merged_Filtered_Data.txt data saved in Section 4.7:
       Time [s] ; Mode 1-1 Frequency [Hz] ; Multimeter Temperature [K] ; Mode
1-1 Voltage [V] ; Mode 1-2 Frequency [Hz] ; Mode 1-2 Voltage [V] ; Fringe
Voltage [V] ; Rolling_Median_Factor = <> ; Trim Start [s]: <> ; Trim End [s]:
<> ; Date and time of saved file
    For the Merged_Filtered_Time_Compensated.txt data saved in Section 5.1:
       Time [s] ; Mode 1-1 Frequency [Hz] ; Multimeter Temperature [K] ; Mode
1-1 Voltage [V] ; Mode 1-2 Frequency [Hz] ; Mode 1-2 Voltage [V] ; Fringe
Voltage [V] ; Rolling_Median_Factor = <> ; Trim Start [s]: <> ; Trim End [s]:
<> ; Date and time of saved file ; Compensated Date: Date and time of saved
file containing compensation time ; Uncompensated Lag = <> ; Compensated Lag
= <>
```

Colourmap Plotter README.txt

```
The goal of this script is the extract results from the data that was merged
and filtered in the "Merging and Filtering.py" script.
This script prompts the user to load the aging from different datasets
recorded by the creator of the script Michel Stephan.
It will then give the user the option to either load the uncompensated
"Merged_Filtered_Data.txt" or the time compensated
"Merged_Filtered_Time_Compensated.txt" data, to extract results for the raw
data of the signals chosen to be streamed.

The following results are produced:
    Raw data plot containing signals of the frequency for mode 1-1 and mode
1-2 as well as the thermistor temperature, all as a function of time
    Raw data plot containing the temperature drift equivalence of mode 1-1
and the thermistor temperature as a function of time
    Colormap plot containing the frequency of the mode 1-1 as a function of
the temperature of the thermistor, with a color gradient to represent the
time dependency

SCRIPT OVERVIEW (Following section labels in script comments):
    A - Loads the required libraries
    B - Class used to correctly format plots by including a custom offset
    C - Function and section used to quickly change the plot sizes, DPI and
border without searching through script
    D - Function used to convert data between different physical quantities
    E - Function used to plot the raw data signals using twin axis
    F - Function used to plot the raw data signals as an equivalence of
temperature drift rather than frequency
    G - Function used to plot the colormap
    1 - Section used to set the text formatting of the script
    2 - Section containing different material properties used in the script
computation
    3 - Section containing different data processing parameters that can be
changed
    4 - Section that prompts the user to choose which data set he wished to
load
    5 - Section that loads the merged and filtered data for the entire
datalog and stores the data in a large dataframe
    6 - Section that determines the appropriate timescale and time label to
use based on the time length of the data
    7 - Section used to send the appropriate information to the Section E and
plot respective information
    8 - Section used to send the appropriate information to the Section F and
plot respective information
    9 - Section used to send the appropriate information to the Section G and
plot respective information

This script is pretty straightforward and is mostly used to extract data
thus, there is not many user dependent options.

NOTE: It is important that the user ensures that the drive label indicated in
the script, which is the "Drive_Label"
      variable in Section 3, matches the actual drive label on the computer
where the datalogs for the stability
      measurements are stored.
```

```
NOTE: If you want to quickly change the dimensions of the plots, simply use
the Section B which is designed to give the user quick access to all of the
graph's sizing settings
```

Frequency Drift Plotter README.txt

```
The goal of this script is the extract results from the data that was merged
and filtered in the "Merging and Filtering.py" script.
This script prompts the user to load the aging from different datasets
recorded by the creator of the script Michel Stephan.
It will then give the user the option to either load the uncompensated
"Merged_Filtered_Data.txt" or the time compensated
"Merged_Filtered_Time_Compensated.txt" data, to extract results for the
frequency drift of the analyzed membrane.

The following results are produced:
    Colormap plot for each datasets with the frequency of the mode 1-1 as a
function of the temperature of the thermistor, with a color gradient to
represent the time dependency
    Temperature compensated frequency drift plots for each datasets which
represent the relative frequency drift of the mode 1-1 as a function of time
    Temperature compensated frequency drift plots of the total dataset
representing the relative frequency drift in ppm for mode 1-1 and mode 1-2 as
a function of time with fitted choice of trend for mode 1-1

SCRIPT OVERVIEW (Following section labels in script comments):
    A - Loads the required libraries
    B - Function and section used to quickly change the plot sizes, DPI and
border without searching through script
    C - Function that determines the appropriate timescale and time label to
use based on the time length of the data
    D - Function used to modelize the double logarithmic trend used in the
fitting function of Section I
    E - Function used to modelize the hybrid logarithmic-linear trend used in
the fitting function of Section I
    F - Function used to plot the individual colormap plot and fit, for every
determined time slice, a linear trend used to predict frequency change as a
function of temperature
    G - Function used to plot the temperature compensated frequency drift
plot for each datasets
    H - Function used to plot the temperature compensated frequency drift
plot for the entire dataset, relay data to Section I
    I - Function used to take the temperature compensated frequency drift
data from Section G, loads the fit guesses from the appropriate texfile, fit
a the appropriate trend model
        from Section D/E while offering a preview, and saves the good fit
parameters in the appropriate textfile.
    1 - Section used to set the text formatting of the script
    2 - Section containing different material properties used in the script
computation
    3 - Section containing different data processing parameters that can be
changed
    4 - Section that prompts the user to choose which data set he wished to
load along other script options
    5 - Section that loads the merged and filtered data for the entire
datalog, stores the data in a large dataframe, gather the linear trend for a
single dataset,
        generates the individual colormap plot with Section E and the
individual frequency drift plot with Section F and computes the mean
compensation linear trend for the whole datalog
```

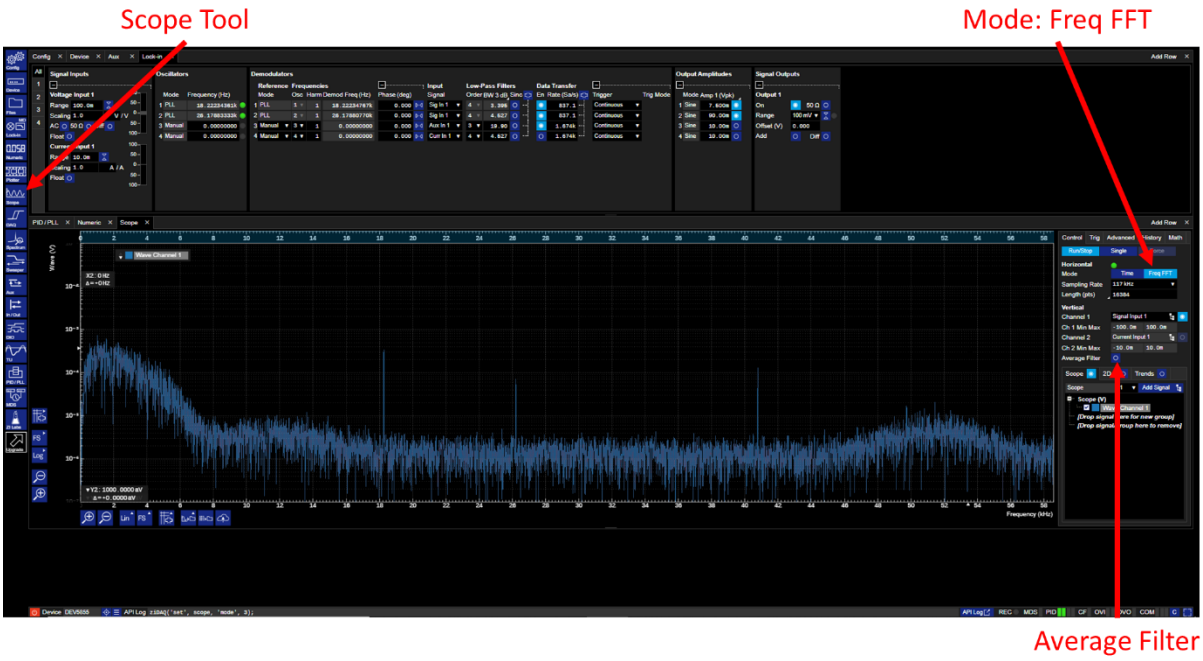
```
    6 - Section that is used to load the fit guesses, attempt fit with
optional filtering if downsampling is required, provide a preview of the
generated trend all using Section H
        and save the good fitting parameters to the textfile so they can be
easily loaded the next time the script is executed

This script is pretty straightforward and is mostly used to extract data
thus, there is not many user dependent options.

NOTE: This scrip tcontained data processing parameters in Section 3 that have
to be changed depending on the situation.

    It is important that the user ensures that the drive label indicated in
the script, which is the "Drive_Label"
        variable in Section 3, matches the actual drive label on the computer
where the datalogs for the stability
        measurements are stored.

    The slice of time taken from each data set to generate a linear trend
used to gather the temperature dependent frequency drift is controlled by the
variable "Averaging_Interval" and is of a default value of 6. This
    means that it will take every 6 hours of the datasets and generate a
linear trend with that slice of time. This interval can be changed by simply
attributing a new value in hours to the variable.

    Due to the effects of the HVAC system, the fitted linear trend will
retain the cyclic upper and lower plateau of data inside the drift plots. To
collapse these two plateau, a slope modifier value must be affected to
    the total slope used to gather the total frequency drift plot. The slope
modifier value is controlled by the "Slope_Modifier_Percentage" and is of a
default value of 8% which works well and is devided by 100 in the script.

NOTE: If you want to quickly change the dimensions of the plots, simply use
the Section B which is designed to give the user quick access to all of the
graph's sizing settings

NOTE: The file containing the recorded fit guesses must be in the same folder
as the script to ensure that it works well. The structure of the double
logarithmic model guess textfile ("Fit_Guesses_Log_Log.txt") is:
       A1 = ; <> ; A2 = ; <> ; t0a = ; <> ; B1 = ; <> ; B2 = ; <> ; t0b = ;
<> ; Additional Filtering Final Rate [Hz] ; <> ; DATE and TIME of the
executed fit

NOTE: The file containing the recorded fit guesses must be in the same folder
as the script to ensure that it works well. The structure of the hybrid
logarithmic-linear model guess textfile ("Fit_Guesses_Log_Lin.txt") is:
       A1 = ; <> ; A2 = ; <> ; t0a = ; <> ; B1 = ; <> ; <> ; t0b = ; <> ;
Additional Filtering Final Rate [Hz] ; <> ; DATE and TIME of the executed fit
```

C. *LabOne* Mode Pinpointing and PLL Setup for Frequency Tracking

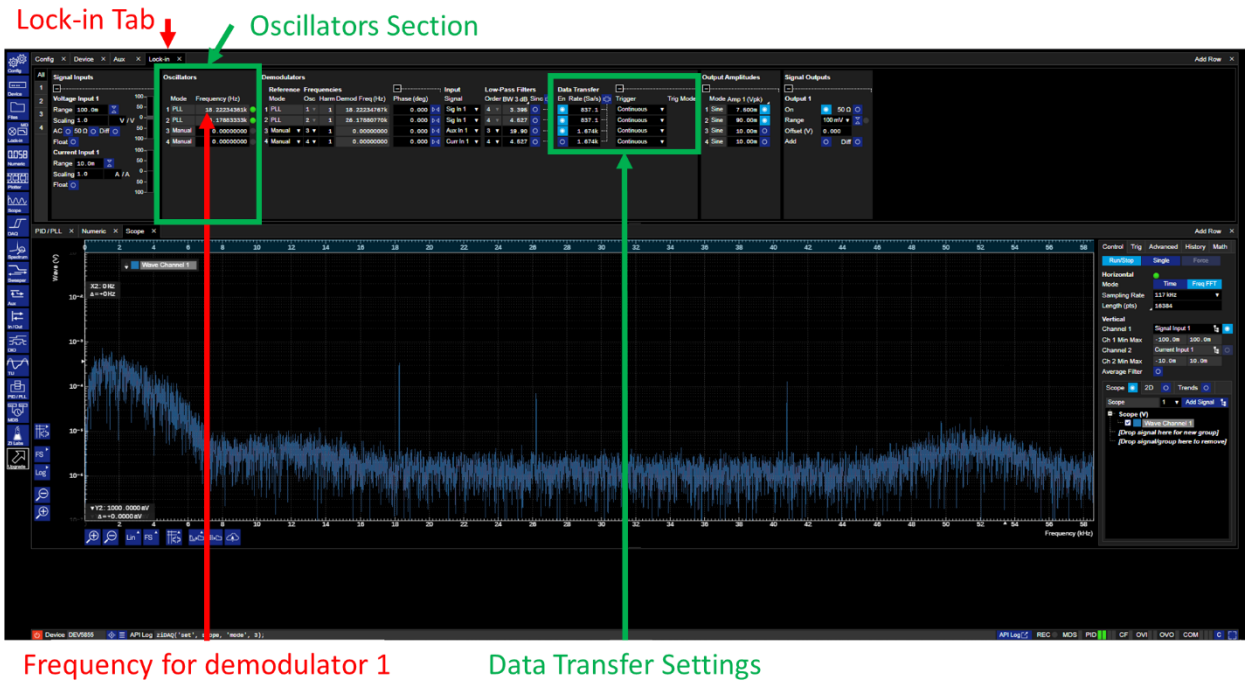
Here is the proper way to set up the PLL assuming that a membrane is resonating, that the optical system to interrogate the membrane is successfully installed and that the fringe has been successfully centred.

1- One the *LabOne* interface, select the *"Scope"* tool and on the option panel to the right, set the "*Mode"* to *"Freq FFT"*.

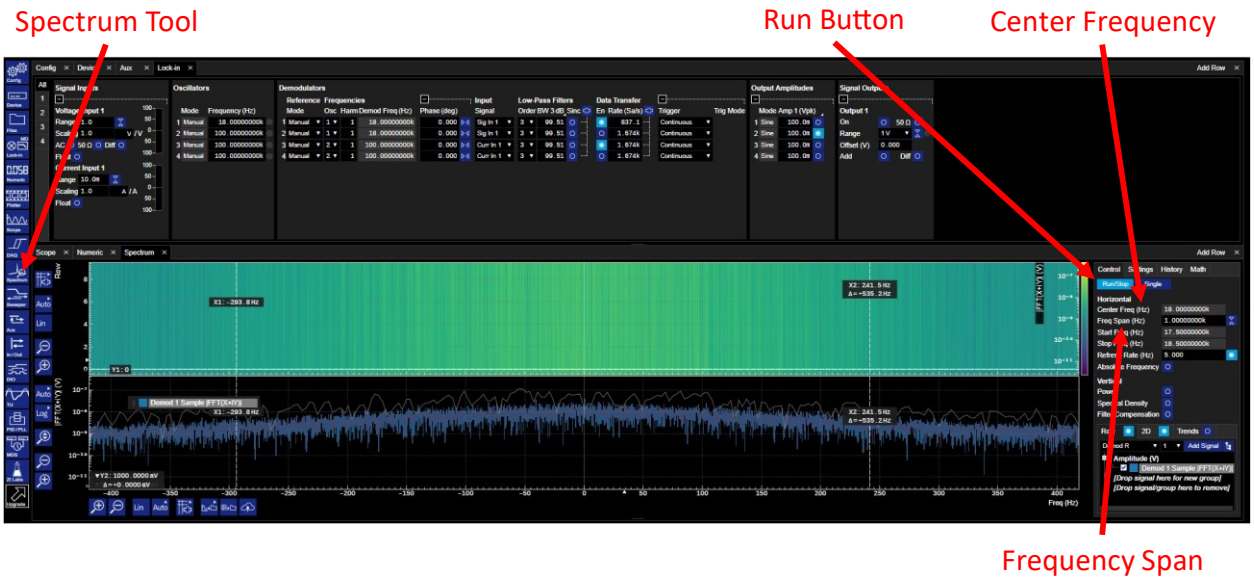
2- Roughly identify the peak frequency that matches the desired mode by actuating mechanically and perturbing the membrane (lightly tapping on the vacuum chamber or cell with a ruler). *"Average Filter"* option can be set at around 10 or 20 to elongate the visual representation of the excited frequency and help the user find the proper peak.



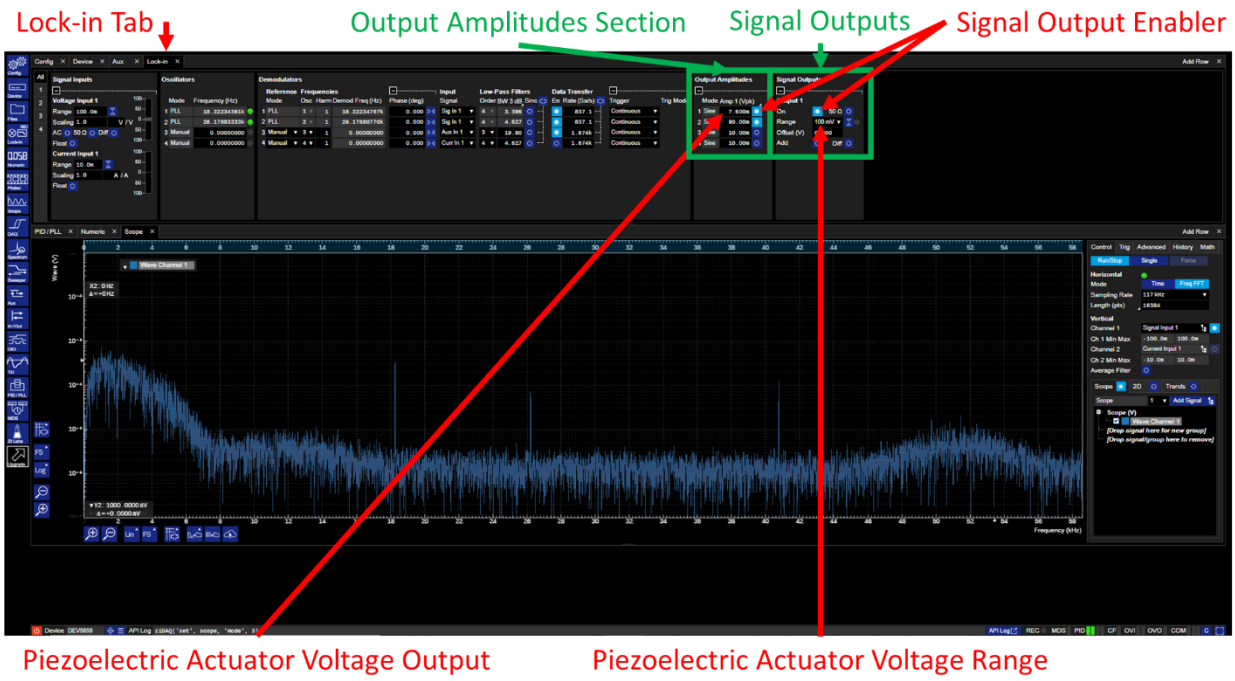
3- In the *"Lock-in"* tab, in the *"Oscillators"* section, input the roughly estimated frequency value of the studied mode found in step 2 for the demodulator 1. Ensure that the data transfer is enabled in continuous mode.



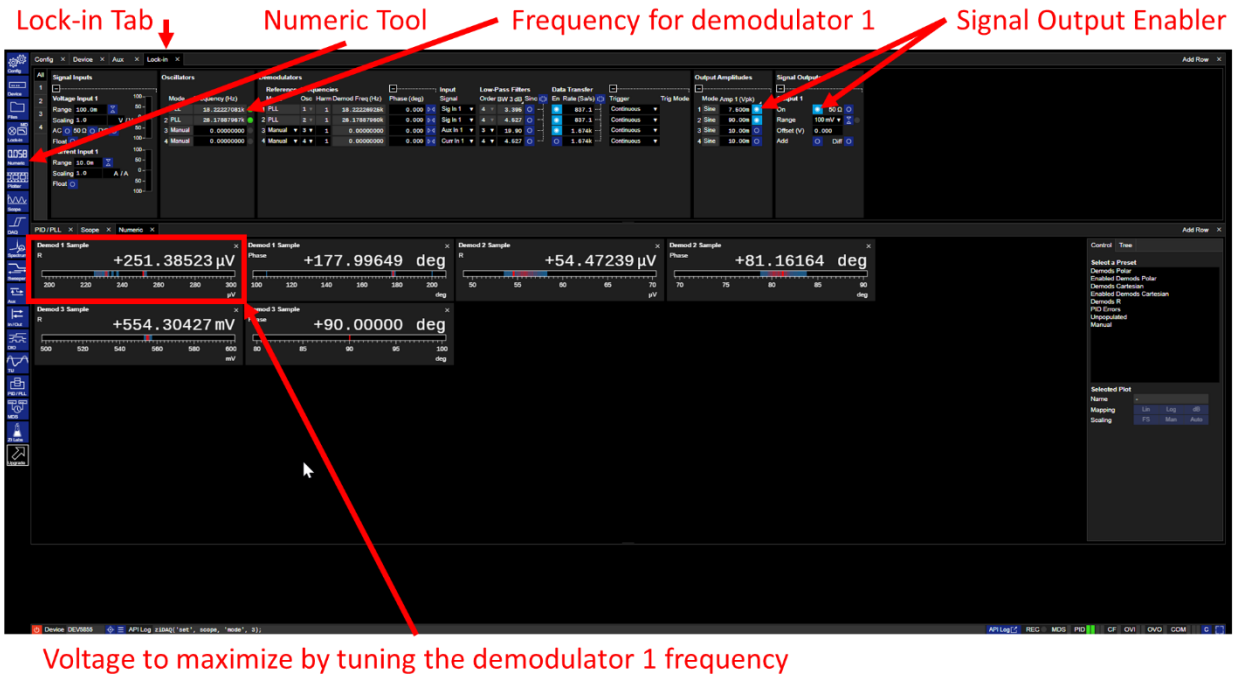
4- Using the "*Spectrum"* tool, set the *"Center Freq (Hz)"* to the estimated value of the studied mode found in step 2, set the "*Freq Span (Hz)"* to 1 kHz, click the "*Run/Stop*" button and observe where a peak is formed in the spectrum. By analyzing the frequency difference between the current set frequency in the demodulator and the peak in the spectrum, modify the frequency value in the *"Lock-in - Oscillators"* section until the peak of the spectrum is centralized at 0 Hz.



5- At this point, you can specify a voltage output for the piezoelectric actuator activated in the *"Output Amplitudes"* section of the *"Lock-in"* tab. Start with a value of 10 mV and ensure both the output of the *"Signal Outputs"* and the *"Output Amplitudes"* are enabled. If you see the spectrum spike largely, you are well positioned. If not you can increase the piezoelectric voltage output to 100 mV. NOTE: Ensure that the range of the signal is no more than 1 order of magnitude larger than the output values you specified.



6- To be positioned even more accurately on the resonance peak, close the piezoelectric actuator's output, open the *"Plotter"* or the *"Numeric"* tool at the bottom of the interface and have the *"Lock-in"* tab opened at the top of the *LabOne* interface, and modify the demodulator 1 frequency by increasing or decreasing by steps of values of 1 (using up or down arrows on the keyboard) the first decimal digit of the frequency until the value that maximizes the voltage of the signal is reached. Use the right arrow on the keyboard to move to the next digit and repeat the process by tuning the digits of the frequency until the signal's voltage is maximized. At this point, re-enable the piezoelectric actuator's output at a starting low value of 10 mV and ramp up towards 100 mV if needed (it is not required, unless necessary, to go beyond 100 mV of actuation). You must now quickly setup the PLL while the demodulator's frequency is still centred on the membrane's actual frequency.



7- Open the "*PID/PLL*" tool on the left of the *LabOne* user interface. Only the settings identified in red, in the supporting image below, must be modified.

   a. In the *"Advisor"* panel situated on the right, start by changing the *"DUT Model"* to the *"Resonator Frequency"* mode.

   b. Specify the resonance frequency of the studied mode by copying and pasting the demodulator 1 frequency that was fine-tuned in step 6.

   c. Input the quality factor of the mode.

   d. Input a *"Targeted bandwidth value"* of around 1 Hz.

   e. Click on the *"Advise button"* and wait for the PID settings to be calculated. The obtained values, for a setting comparable to the one presented in this thesis, should be close in order of magnitude to the ones showcased in the image below. If the *"BW (Hz)"* and *"PM (deg)"* lights at the bottom of the *"Advisor"* panel are in green, proceed to the next step. If the lights are red, restart at step d by

inputting a *"Targeted bandwidth value"* that is closer to the value shown in the *"BW (Hz)"* slot.

f. When the *"BW (Hz)"* and *"PM (deg)"* lights at the bottom of the panel are in green, proceed by pressing *"To PID"* to send the parameters to the PID panel.
g. In the *"PID/PLL"* panel situated on the left, begin by setting the *"Tracking mode"* to the *"PLL"* option.
h. Set the *"Tracking adjustment mode"* to the *"PID Coeff"* setting.
i. Ensure that the *"PLL input selector"* is set on the right demodulator phase, which is the demodulator 1 in the case of this example.
j. Ensure that the *"PLL output selector"* is set on the right demodulator frequency, which is the demodulator 1 in the case of this example.
k. Copy and paste the frequency demodulator 1's in the *"Oscillators"* section that was fine-tuned in step 6 into the *"PLL center frequency"* slot.
l. Put the *"PLL limits"* as values, for the lower and upper limit respectively, of -100 and 100 Hz. If the PLL has a hard time tracking the frequency, you can try to play with these limits.
m. Enable the PLL using the "*PID enabled button"* at the top of this panel.

At this point, the PLL should take over and the *"Error (deg)"* value should start at a large value and slowly trend downwards towards +/- 5. Under a value of 5, the PLL is working and will successfully track the frequency of resonance, while the PLL indicator light will turn green. An error value of sub 1 is good to give enough room for the PLL to drift and ensure that it won't lose its lock. If the PLL did not manage to successfully track the membrane, close the *"PID*

enabler button", close the piezoelectric actuator output, fine-tune the frequency as shown in step 6 once more, re-enable the piezoelectric actuator output, copy and paste the new fine-tuned frequency to the *"PLL center frequency"* and enable one again the *"PID enabler button"*. One the lock is established; you can open the *"Numeric"* tool at the top and have the *"PID/PLL"* tool at the bottom. You can then carefully modify the *"Phase modifier"* value by steps of 5 degrees in any direction until the voltage of the demodulator 1's signal is maximized which will thus allow you to have the most stable PLL.

D. Undergrad Thesis: Athermalization of SiN Membrane Resonator

Foreword: We previously conducted theoretical study, on the stress variation in our SiN devices caused by the difference of thermal expansion between their substrates and their membranes. The developed principle of athermalization consisted of bonding a plate, of specific geometry and material properties, to our substrate in order to equilibrate the thermal expansion coefficients. This would, theoretically, eliminate the thermal expansion stresses affecting our membranes. This methodology was not implemented in the current aging study, but we have used the knowledge we gained from this work to carefully design, in this work, a new mounting method for our membranes (magnetic mounting) which would eliminate some stresses linked to thermal expansion. The initial idea of athermalization is included here for archival and availability purposes.

Athermalization of Silicon Nitride Membrane Resonator

by Michel Stephan

MCG4100
Thesis Department of Mechanical Engineering
University of Ottawa

April 2021

Thesis Supervisor: Dr. Raphael St-Gelais

Abstract

This thesis explores a simple athermalization method for neutralizing unwanted frequency fluctuations, due to thermal drift, in a silicon nitride membrane used to sense radiation. The developed method relies on attachment of a low thermal expansion mount below the silicon substrate on which the membrane is fabricated. This mount creates bending in the structure upon thermal expansion. With the presence of this new layer and through the bending mechanics of a layered structure, a compensation stress composed ofa bending and axial component will be created in order to neutralize the unwanted stress issued from the thermal drift of the membrane. With the help of a developed python code that uses equations taken from [1], the mount is designed with a specific thickness, coefficient of thermal expansion and modulus of elasticity to provide adequate athermalization. It has been determined that the optimal mount is made from invar and would have a thickness of 2.04 mm to completely neutralize the undesired stress sources.

Table of Contents

List of Figures

List of Tables

1) Introduction

Silicon nitride (SiN) nanomechanical resonators are used for sensing a wide variety of physical quantities, such as forces [2], temperature [3], or radiation [4]. For force sensing, silicon nitride membranes are used for their high mechanical quality factor (i.e., low mechanical dissipation). A high quality factor ensures a low frequency and low amplitude fluctuation which will result in a high-precision measurement. High Q-factor isachieved through SiN's large aspect ratio and high stress [5]. In [2], a membrane is used as a force transducer that vibrates under a scanning force microscopy device. The device will gradually position its scanning tip closer to the membrane until a shift in the resonance frequency is obtained, from which the force can be quantified. A second typical application measures temperature through the frequency analysis of silicon nitride nanostrings [3]. In this case, SiN is used because of its high Q-factor and high sensitivity to temperature. Measurements are taken by analyzing the resonating frequency of the string which will decrease as a function of its temperature. Thus, by knowing the mechanical properties of SiN and of the substrate of the string, the measured frequency drift will be used to quantify temperature.

The application on which our lab focuses is radiation sensing through resonance frequency measurement of nanomechanical resonators. In this case, thermal radiation is detected by measuring a shift of the mechanical resonance frequency upon radiation absorption [6]. Other important work in these directions have been realized by Schmid [7] as well as Roukes [8]. Recent experiments use SiN for thermal radiation sensing because this material provides a high Q-factor. Here high Q-factor means a very narrow resonance,

which is easier to detect and pinpoint. Thus, with SiN we can create sensing devices with high sensitivity to temperature, high temperature resolution and radiation-dominated thermal coupling with the environment [6]. When heat is transferred to the membrane through exposure to radiation, its properties change. Heat causes an expansion of the SiN membrane, thus lowering its built-in tensile stress and causing a detectable frequency shift.

A common problem with frequency shift sensing techniques is frequency drift. This phenomenon occurs when an interfering source induces an undesired stress on the resonator's structure, which results in an unwanted resonance frequency change interfering with the measured signal (see **Figure 1**). In the case of radiation sensors, a common cause of drift is thermal expansion of the substrate that generates an opposing stress inducing a frequency drift, as schematized in **Figure 1**.

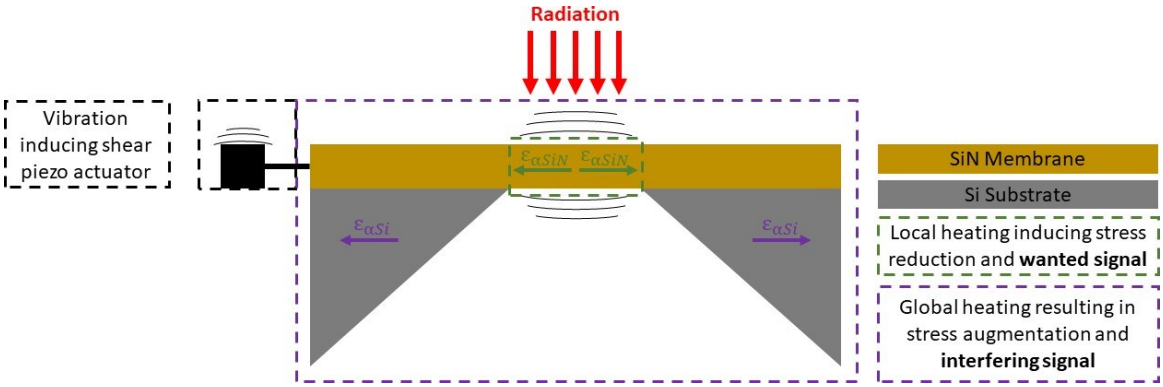


Figure 1: Opposing effect of wanted stress due to local heating of SiN membrane and interfering stress due to Si substrate heating

An existing solution for alleviating this effect is to implement a tuning fork with temperature compensation [9]. As it can be observed in **Figure 2**, the nanobeam resonator tuning fork is composed of two parallel cantilever beams that will be clamped by a Y-shaped thermal compensation device. The compensation device, which is much bigger than

the resonator itself (i.e., the blue square in **Figure 2**), is composed of two clamping nonlinear springs paired with a doubly clamped tension bar that is attached to the resonator [9]. This passive method applies a modifiable predetermined amount of stress to the membrane to exceed and counter its residual stress. As thermal expansion occurs, the tension bar will deform and adjust the stress on the nanobeam resonator. The continuous expansion of the bar will eventually pull on the non-linear springs. They will generate an opposing stress ensuring that the tension bar does not overcompensate the thermal stress effects on the resonator. The main drawback of this method is that the integration of the compensation device is complexified since it is much bigger than the resonator itself (i.e.,the blue rectangle in **Figure 2a**.

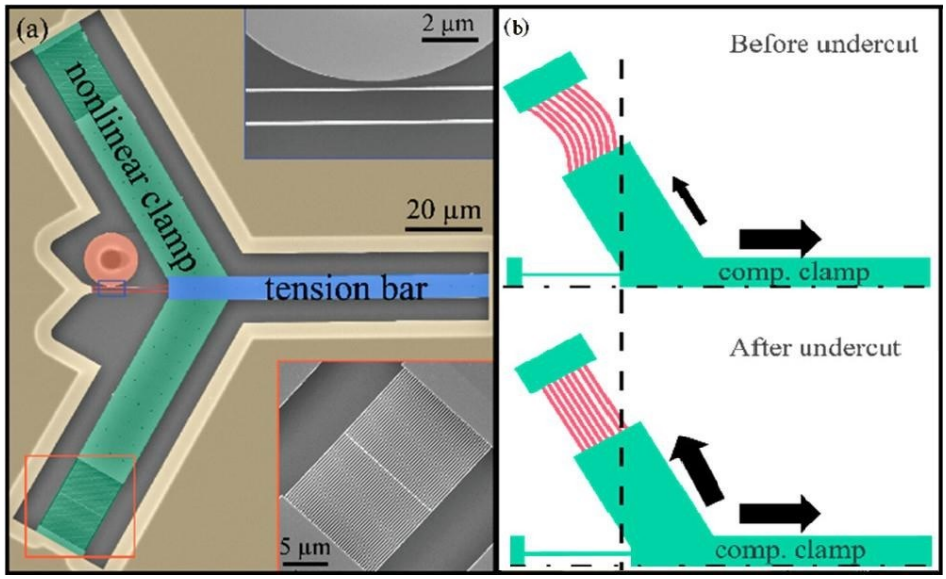


Figure 2: Thermal compensating tuning fork with microdisk optical resonator read-out coupled to nanobeam resonator (in blue rectangle). Compensation device composed of non-linear springs (in red rectangle) and tension bar. *[9]*

Here we investigate a simple and compact athermalization method based on stress compensation achieved by adding a backing layer of different coefficient of thermal

expansion (CTE). With the proper CTE, this layer cancels the undesired stress induced by temperature drifts in the substrate [10].

We note that athermalization is not limited to the field of nanomechanical sensors. More generally, in other fields, athermalization is used in optics to cancel the effect of temperature drift that influences optical properties such as the refractive index [11], thus leading to the modification of frequency readings by optical resonant sensors [12].

2) Theory

2.1) Membrane Stress Equation

The natural resonance frequency (*f,* in Hz) of a freestanding rectangular SiN membranes depends on the total tensile stress (σ_T, in Pa), on dimensions (L_x, L_y in m), on resonance modes orders (denoted by integer n, j), and on SiN material density (ρ = 2.9 g/cm^3 [13]). For a square membrane, the length and width of the membranes are equal ($L_x = L_y = L$), thus reducing the equation to [5]:

$$f = \pi\sqrt{j^2 + n^2}\frac{1}{L}\sqrt{\frac{\sigma_T}{\rho}}. \qquad (\mathbf{1})$$

The membrane stress is affected in multiple ways by temperature. Local heating of the membrane itself (caused by incoming radiation in our case [4]) is the detected signal, which is desired. As schematized in **Figure 1**, radiation absorbed by the membrane will cause it to expand and relax its built-in stress, thus lowering its frequency, as shown by equation **(1)**. Here we consider local changes in geometry and density (i.e., changes in L and ρ) to be negligible. Since the heat source is applied over the entire system, the substrate will also heat up and expand. This expansion will stretch out the membrane, thus inducing an opposing variation of the membrane stress. Due to the relatively large thermal mass of the substrate, this phenomenon is slower and shows up as a parasitic slow drift during radiation sensing experiments.

More quantitatively, the temperature-dependant membrane stress (σ_T) is given by

$$\sigma_T = \sigma_{0_{SiN}} - E'_{SiN} \cdot \alpha_{SiN} \cdot (T_m - T_0) + E'_{SiN} \cdot \alpha_{Si} \cdot (T_{subs} - T_0). \qquad (\mathbf{2})$$

which was originally derived in the context of a string-based temperature sensor [3]. In

equation **(2)**, we use the modified modulus of elasticity ($E' = \frac{E}{1-\nu}$, where ν is the Poisson ratio) to account for biaxial stresses in a two-dimensional membrane. In **(2)**, the first term ($\sigma_{0_{SiN}}$) represents the formation stress of the membrane which is of 100 MPa. The second term is the stress variation due to the local thermal expansion of the SiN membrane relative to an initial reference temperature (T_0), typically room temperature. E'_{SiN} is the modified biaxial stress modulus of elasticity for the membrane, α_{SiN} is its coefficient of thermal expansion (CTE), and T_m is the temperature of the membrane. The third term represents the opposed stress issued from the expansion of the Si substrate. Similarly to the second term, it depends on the initial temperature, the temperature of the substrate (T_{subs}), the membrane's modified biaxial stress modulus of elasticity (E'_{SiN}) and the CTE of the substrate (α_{Si}). The third term uses the modified modulus of elasticity of the membrane. This result originates from the assumption that the membrane, being very thin, will follow exactly the substrate deformation [3]. The second term of equation **(2)** is the desired signal while the third one, is the drift term that must be suppressed.

Our approach is to artificially create a situation where the CTEs in **(2)** are balanced,by introducing a third layer that will induce a neutralizing stress. If $\alpha_{Si} = \alpha_{SiN}$, equation **(2)** reduces to:

$$\sigma_T = \sigma_{0_{SiN}} - E'_{SiN} \cdot \alpha_{SiN} \cdot (T_m - T_{subs}). \qquad (3)$$

In this case, membrane temperature variations (T_m) are always measured relative to the substrate, thus cancelling its slow drift effect on the measured signal. In other words, if the membrane temperature (T_m) and the substrate temperature (T_{subs}) vary together due to slow ambient temperature drift, the detected membrane stress will remain unchanged.

Only local variations of the temperature, affecting the membrane only, will produce a detectable frequency shift.

Carefully selecting a material with a lower CTE than the substrate is essential in ensuring that the bending stress term opposes the undesired stress term. This will lead to the successful athermalization of the system. Our approach is largely inspired by the workof T.W. Clyne on bending stress [1], which we adapt here to the specific case of silicon substrates with SiN membranes.

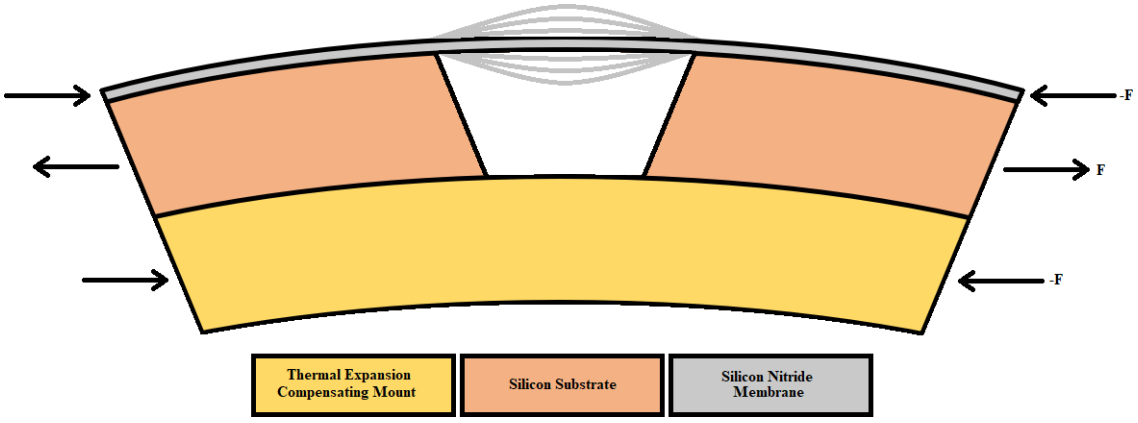


Figure 3: Athermalizing structure for SiN membrane

By adding a third layer to the assembly (i.e. the "mount" in **Figure 3**), the stress equation **(2)** will include an additional term (σ_C), which accounts for the mount's CTE, aswell as the effect of bending (see **Figure 3**). Our approach is to adjust σ_c to cancel out unwanted linear thermal expansion stresses.

$$\sigma_T = \sigma_C + \sigma_{0_{SiN}} - E'_{SiN} \cdot \alpha_{SiN} \cdot (T_m - T_0) + E'_{SiN} \cdot \alpha_{Si} \cdot (T_{subs} - T_0). \qquad (4)$$

The compensating stress term (σ_C) can be expanded to explain the way the stresses are formed when a structure of two-layered materials is subjected to thermal expansion.

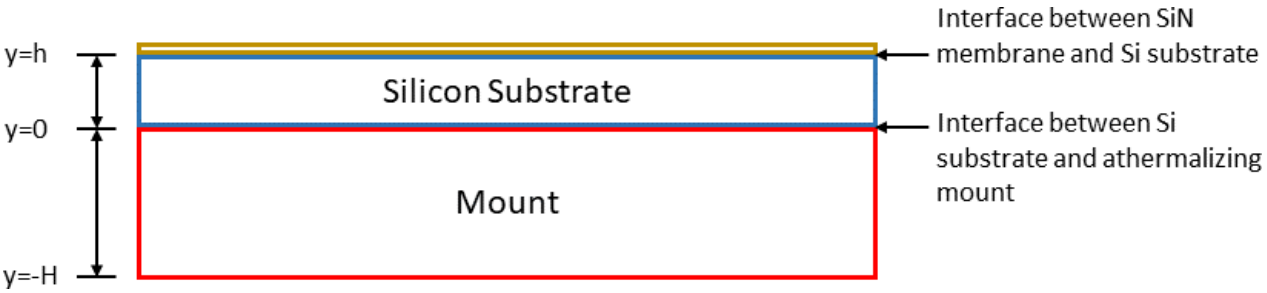


Figure 4: Layout of the different layers of resonator with athermalizing mount

To find the compensating stresses in the SiN membrane, one must use the equations computing the residual stresses of interfacial debonding occurring at the interface of interest. In our case, as observed in **Figure 4**, we must evaluate the stresses at the SiN membrane and the Si substrate interface (y=h), to be able to develop an appropriate athermalizing mount.

We begin by finding the misfit strain issued from the thermal expansion due to heating of the device [1]. We suppose that the temperature changes in the membranes (T_m) are equal to the ones in the substrate and the mount. To find the misfit strain, the following equation is used: $\Delta\varepsilon = \int_{T_0}^{T_{subs}} \big(\alpha_C(T) - \alpha_{Si}(T)\big) dT$ [1]. If the CTEs are constant, this reduces to:

$$\Delta\varepsilon = (\alpha_C - \alpha_{Si}) \cdot (T_{subs} - T_0), \tag{5}$$

where α_C is the CTE of the thermal expansion compensating mount.

Once the substrate and mount thermally expand, the structure will bend resultingin the creation of a neutral axis. The next parameter that must be calculated is δ, the difference between the central interface of the resonator and the neutral axis. We define

the central interface here as the interface between the silicon substrate and the mount.

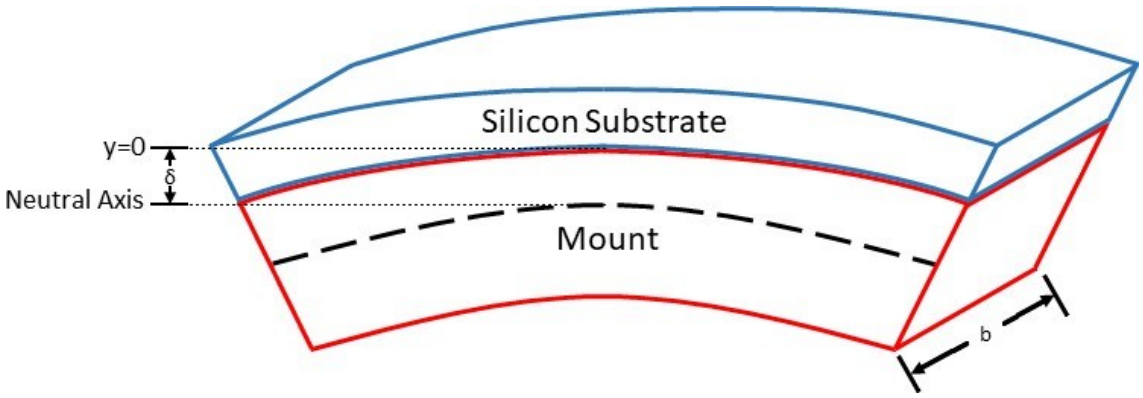


Figure 5: Representation of the bent resonator with athermalizing mount and neutral axis

The following equation is used to find δ which depends on the geometrical factor presented in **Figure 4** and **5** as well as the properties of the different materials [1]:

$$\delta = \frac{h^2 E'_{Si} - H^2 E'_C}{2(hE'_{Si} - HE'_C)}. \quad (6)$$

In equation **(6)**, h is the thickness of the silicon substrate which is also the distance between the substrate-membrane and substrate-mount interface. H is the thickness of the mount. The modified biaxial stress modulus of elasticity for the mount is represented by E'_C while the one for the substrate is E'_{Si}.

The thermal expansion misfit strain induces forces (P) creating a bending moment (M) on the beam as observed in **Figure 6**. To compute the compensating stress required to athermalize the resonator, another element to be determined is the ratio between the bending forces (P) as well as the mount and substrate width (b) defined in **Figure 5**.

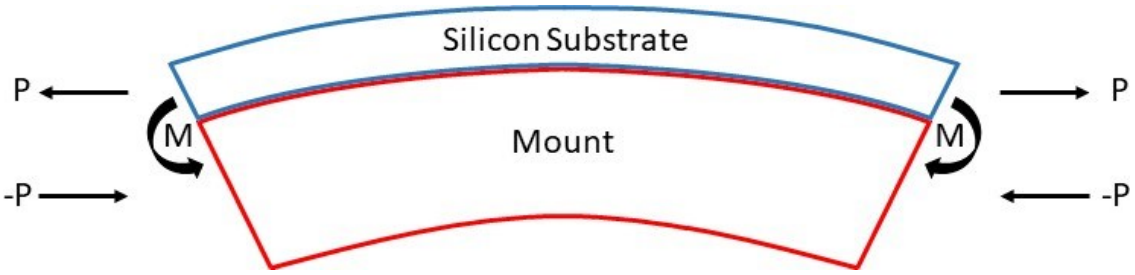


Figure 6: Resonator's bending moment and forces

The ratio $\frac{P}{b}$ is given by the following relation [1]:

$$\frac{P}{b} = \Delta\varepsilon\left(\frac{hE'_{Si}HE'_C}{hE'_{Si} + HE'_C}\right). \tag{7}$$

The last parameter that needs to be computed in order to find the compensation strain is the curvature of the bent structure which is given by [1]:

$$k = \frac{6E'_{Si}E'_C(h+H)hH\Delta\varepsilon}{{E'_{Si}}^2h^4 + 4E'_{Si}E'_Ch^3H + 6E'_{Si}E'_Ch^2H^2 + 4E'_CE'_{Si}H^3h + {E'_C}^2H^4}. \tag{8}$$

With the aforementioned parameters found, the compensation strain (ε_C) due to the implementation of the third layer in the resonator's structure can be computed. Since the interface of interest is the one between the SiN membrane and the Si substrate, we focus on the strain at the top layer of silicon (i.e., at $y=h$) [1]:

$$\varepsilon_C|_{y=h} = \frac{\frac{-P}{bh} + E'_{Si}k(h-\delta)}{E'_{Si}}. \tag{9}$$

It is important to note that $\frac{-P}{bh}$ is the axial compensation stress imposed by the thermal expansion while $E'_{Si}k(h-\delta)$ represents the bending compensation stress occurring in the layered structure.

Knowing that the deformation of the silicon substrate will be the same as the silicon nitride membrane at the evaluated interface, the thermal expansion compensation stress (σ_C) can finally be evaluated using Hooke's law:

$$\sigma_C = \varepsilon_C E'_{SiN}. \quad (\mathbf{10})$$

Plugging in the new expression for σ_C into equation **(4)**, we obtain the following equation representing the membrane's stress:

$$\sigma_T = \frac{\frac{-P}{bh} + E'_{Si}k(h-\delta)}{E'_{Si}} \cdot E'_{SiN} + \sigma_{0_{SiN}} - E'_{SiN} \cdot \alpha_{SiN} \cdot (T_m - T_0) + E'_{SiN} \cdot \alpha_{Si} \cdot (T_{subs} - T_0). \quad (\mathbf{11})$$

2.2) Condition of Athermalization

To clearly showcase the condition under which athermalization occurs, equation **(11)** will undergo some manipulations and simplifications. Firstly, equation **(7)** is modified by substituting equation **(5)** and changing the notation:

$$\frac{P}{b} = (\alpha_C - \alpha_{Si}) \cdot (T_{subs} - T_0), \quad (\mathbf{12})$$

where $\left(\frac{P}{b}\right)_{\Delta\varepsilon} = \left(\frac{hE'_{Si}HE'_C}{hE'_{Si}+HE'_C}\right)$.

Equation **(8)** is also modified by substituting equation **(5)** and changing thenotation. It is rewritten as follow:

$$k = k_{\Delta\varepsilon} \cdot (\alpha_C - \alpha_{Si}) \cdot (T_{subs} - T_0), \quad (\mathbf{13})$$

where $k_{\Delta\varepsilon} = \frac{6E'_{Si}E'_C(h+H)hH}{{E'_{Si}}^2h^4+4E'_{Si}E'_Ch^3H+6E'_{Si}E'_Ch^2H^2+4E'_CE'_{Si}H^3h+{E'_C}^2H^4}$

The equation **(11)** is simplified using equations **(12)** and **(13)**. Algebraic manipulations allow for equation **(11)** to be rewritten as follows:

$$\sigma_T = \sigma_{0_{SiN}} - E'_{SiN} \cdot \alpha_{SiN} \cdot (T_m - T_0) + E'_{SiN} \cdot (T_{subs} - T_0) \cdot \left((\alpha_{Si}) + \frac{-\left(\frac{P}{b}\right)_{\Delta\varepsilon} \cdot \frac{1}{h} + E'_{Si} k_{\Delta\varepsilon} (h - \delta)}{E'_{Si}} \cdot (\alpha_C - \alpha_{Si}) \right). \quad (\mathbf{14})$$

A simple analysis of the term $\frac{-\left(\frac{P}{b}\right)_{\Delta\varepsilon} \cdot \frac{1}{h} + E'_{Si} k_{\Delta\varepsilon} (h-\delta)}{E'_{Si}}$ reveals that it can be reduced to a constant which we will label as the compensating constant (C). Thus, we obtain the simplified final membrane stress equation:

$$\sigma_T = \sigma_{0_{SiN}} + E'_{SiN} \left(\left(\alpha_{Si} + C \cdot (\alpha_C - \alpha_{Si}) \right) \cdot (T_{subs} - T_0) - \alpha_{SiN} \cdot (T_m - T_0) \right). \quad (\mathbf{15})$$

Supposing the temperature of the membrane (T_m) and substrate (T_{subs}) are equal, we conclude that the athermalization of the membrane is achieved once the following condition is met:

$$\alpha_{SiN} = \alpha_{Si} + C \cdot (\alpha_C - \alpha_{Si}). \quad (\mathbf{16})$$

Knowing that $\alpha_{Si} < \alpha_{SiN}$ ($\alpha_{Si} = 2.6\text{x}10^{-6}$ and $\alpha_{SiN} = 2.3\text{x}10^{-6}$ [14]), we can conclude that a negative term multiplied by the compensating constant is required to reduce the right-hand side's CTE so it is equivalent to the SiN's CTE. This ultimately means that the CTE of the mount must also be smaller than that of the Si substrate.

The athermalized system can be represented by the following equation:

$$\sigma_T = \sigma_{0_{SiN}}, \quad (\mathbf{17})$$

where $\sigma_{0_{SiN}} = 100$ MPa.

3) Python Code

We developed a Python model that computes equation **(11)** to automatically determine the stress in the membrane as a function of the designed mount. Material parameters considered in the developed code are given in **Table 1**. Data was taken from peer-reviewed sources whenever possible and, alternatively, from specialized websites or simulation softwares.

Table 1: Properties of selected materials relevant to mount's athermalizing performance

Material	**CTE** $\boldsymbol{\alpha}$ **[1/K]**	**Young's Modulus** $\boldsymbol{E}$ **[GPa]**	**Poisson's Ratio** $\boldsymbol{v}$
Silicon [14]	$2.6\text{x}10^{-6}$	170.0	0.28
Silicon Nitride [14]	$2.\ 3\text{x}10^{-6}$	250.0	0.23
Invar [14]	$1.3\text{x}10^{-6}$	137.0	0.29
ZERODUR® [15]	$5.0\text{x}10^{-8}$	90.3	0.24
Astrositall® [16]	$1.0\text{x}10^{-7}$	92.0	0.28
Fused Quartz [17]	$5.5\text{x}10^{-7}$	72.0	0.17

3.1) Material Properties' Effect on Athermalization

The first test was conducted to study the membrane stress variation with a temperature difference of 100K, as a function of the CTE of the mount (see **Figure 7**).

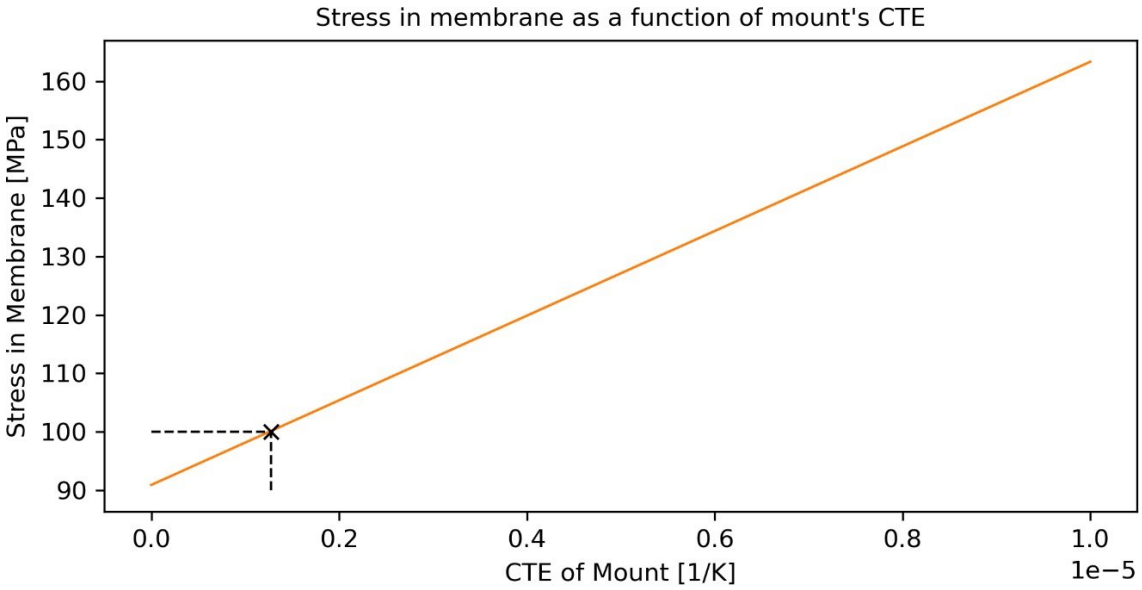


Figure 7: Stress in membrane as a function of mount's CTE, for an assembly at 100K above room temperature. The mount thickness considered is 2 mm. The initial room temperature stress ($\sigma_{0_{SiN}}$ in equation **(11)**) considered here is 100 MPa. This mount athermalizes the membrane for a CTE of $1.274\text{x}10^{-6}\,\frac{1}{\text{K}}$.

The code was executed for the temperature difference between 300 and 400℃, fora 2 mm thick mount with a modulus of elasticity of 137 GPa. The importance of this graph lies in its trend rather than in the data itself. The values obtained depend on many unoptimized variables at this stage. As seen in **Figure 7**, a smaller CTE ensures a lower stress in the membrane. An increase in CTE of the mount will result in a linear increase of the stress. This suggests that we must prioritize materials with low CTE in order to have a good athermalizing mount.

The second conducted test plots the membrane stress as a function of the modulus of elasticity of the mount. The code was executed for the temperature difference between 300 and 400℃, for a 2 mm thick mount with a CTE of $1.3\text{x}10^{-6}\,\frac{1}{\text{K}}$.

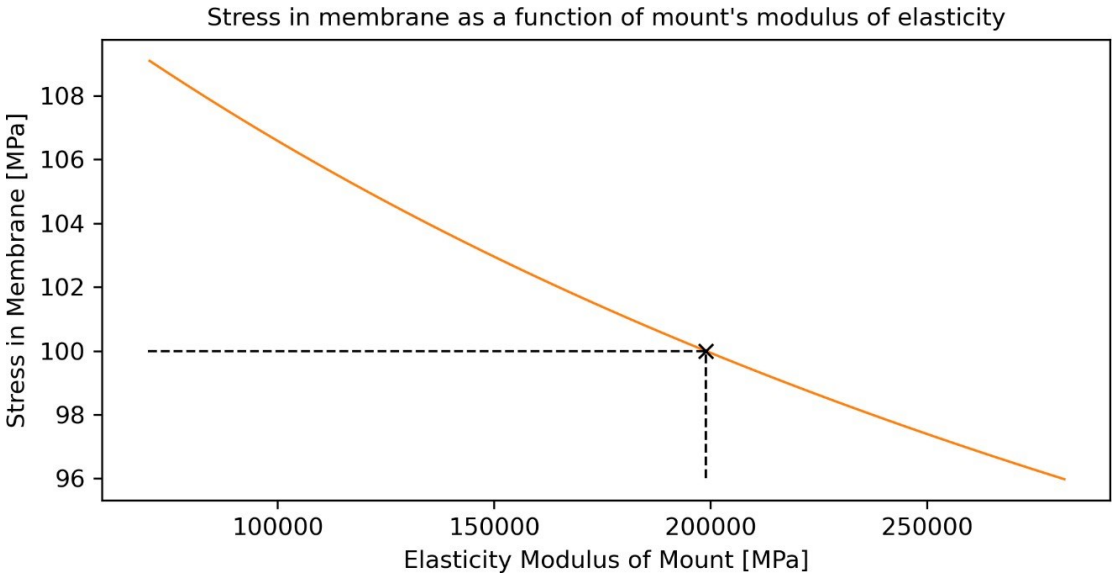


Figure 8: Stress in membrane as a function of mount's modulus of elasticity, for an assembly at 100K above room temperature. The mount thickness considered is 2 mm. The initial room temperature stress ($\sigma_{0_{SiN}}$ in equation **(11)**) considered here is 100 MPa. This mount athermalizes the membrane for a modulus of elasticity of 198.9 GPa.

The plot in **Figure 8** demonstrates a trend that clearly depicts a decrease in the membrane's stress as the modulus of elasticity increases. While a higher modulus of elasticity will contribute to lowering the stress, it does not have as much of an impact as the variation of the CTE. The results obtained suggest that while selecting a material with a higher modulus of elasticity is beneficial, it should not be prioritized over a low CTE.

3.2) Athermalizing Materials

Knowing which properties must be prioritized, we selected 4 materials that would be suitable for our athermalizing mount:

- Invar
- Class 2 ZERODUR®
- Astrositall®
- Fused Quartz

Invar is an alloy composed of 36% nickel and 64% iron. It is characterized by the minimum CTE for any iron and nickel alloy, as observed in the **Figure 9** below. This alloy is known to have one of the smallest CTE among all metals [18].

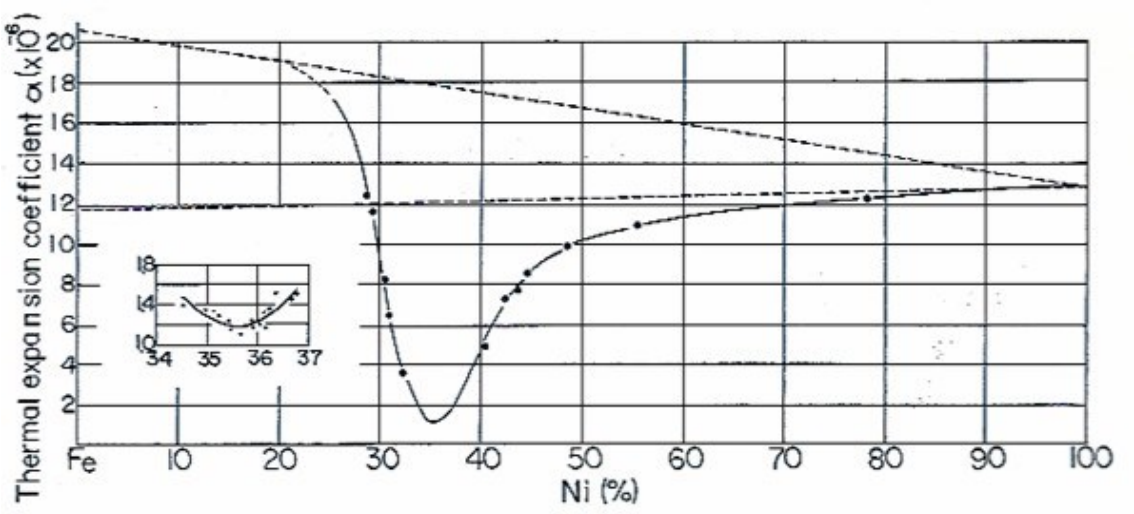


Figure 9: CTE coefficient of Fe-Ni alloy as a function of percentage of nickel *[18]*

The next category of materials that was explored is glass-ceramic composites. Developed mainly for use in optical systems requiring extremely low CTEs such as telescopes, ZERODUR® is a lithium-alumino silicate developped by Schott [15] while Astrositall® is its soviet counterpart [19]. While these materials have smaller CTEs than invar, their modulus of elasticity is also smaller. The last selected material which can be easily manufactured due to the natural abundance of crystalline silica is fused quartz. It can easily be manufactured due to the natural abundance of crystalline silica which, comparatively to other materials, has a low modulus of elasticity [17]. **Table 1** summarizes the materials' properties relevant to their athermalization potential.

3.3) Geometrical Dimensions' Effect on Athermalization

The next step for the development of the optimal mount was to test the geometrical variable—we therefore focus on the impact of the thickness of the mount (H) on the athermalization of the system. In **Figure 10**, rather than plotting the total stress in the membrane (σ_T), only the compensating stress term (σ_C in equation **(10)**) is plotted as a function of the mount's thickness in order to illustrate a counterintuitive behaviour. A curve was generated for each of the selected materials using the properties presented in **Table 1**, still for a temperature difference of 100 K. As observed in **Figure 10**, σ_C initially increases until the mount is of a critical thickness, where it will start to go down drastically. The peak depends on the modulus of elasticity of the material. The material with the highest modulus of elasticity will peak at a smaller thickness while the material with the smallest modulus of elasticity will peak a greater thickness.

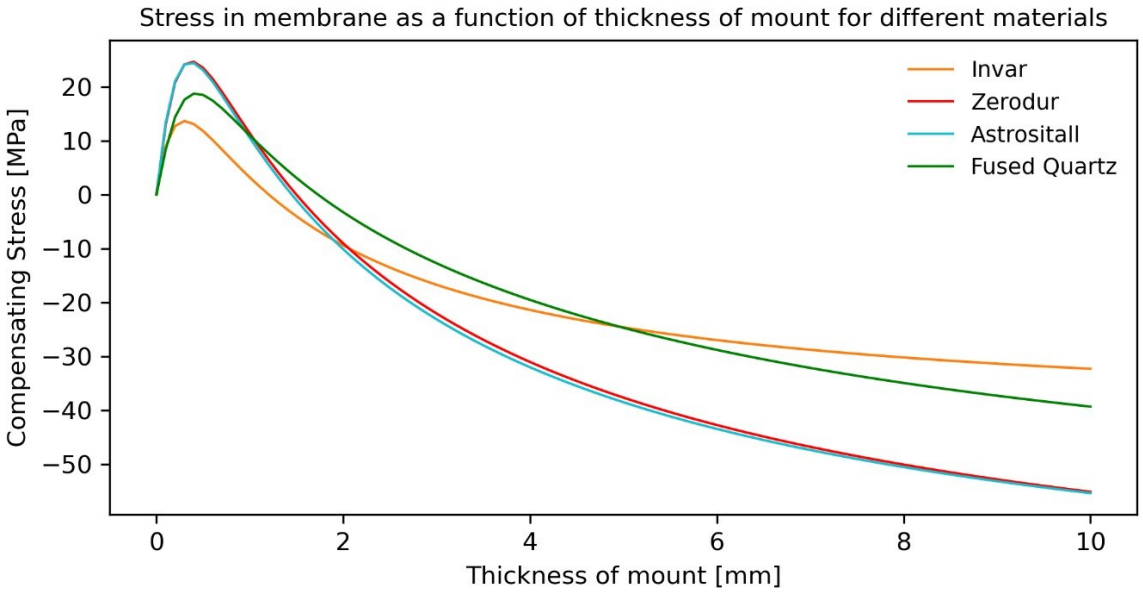


Figure 10: Compensating stress as a function of mount's thickness for selected materials

The presence of this peak reveals that there is a minimal thickness where the mount, depending on its material, begins to have a compensating effect. Whether the mount will have a compensating or contributing stress depends on the curvature of the layered structure which is directly affected by the height of the mount. Since our goal is to neutralize the drift stress, we need a height value that will produce a negative stress.

Figure 10 clearly depicts how the impact of the materials' CTE overshadows the modulus of elasticity when it comes to athermalization performances. While invar noticeably has the weakest athermalizing effect as the thickness of the mount increases, it does have the strongest effect for smaller thicknesses. Another important point to consider in the design of the mount is to have it be as compact as possible so it can easily be implemeted. While selecting a thicker mount would be easier, it would defeat the purpose of having a compact athermalizing method. This is why we opt to select a material that maximises the athermilizing potential while retaining a reasonable thickness.

Selecting a thickness of 2 mm, the stress in the membrane has been plotted for the different materials as a function of the temperature difference of a 100K that the membrane would be subjected to. **Table 1** has been used for the properties of each materials.

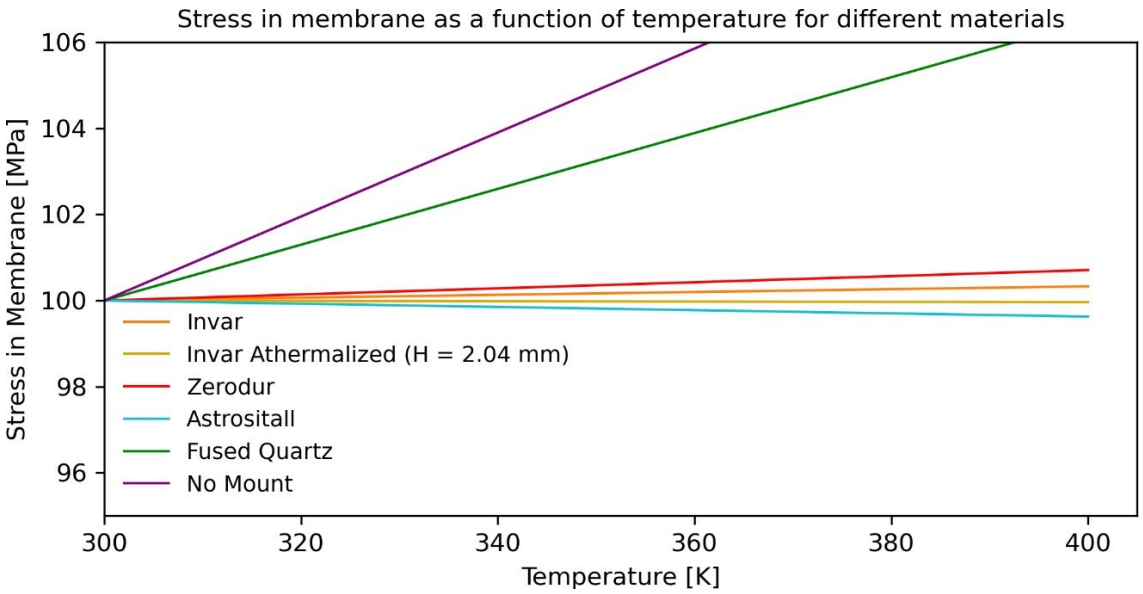


Figure 11: Stress in membrane as a function of temperature for different materials

When comparing the obtained curves to the athermalized one ("Invar Athermalized"), it can be easily noticed that for the same thickness, invar, ZERODUR® and Astrositall® have very good stress compensating effects. While fused quartz is not as performant as the other materials, it does provide some athermalization over the non-athermalized system with no mount. Since the glass-ceramics and the metal have very similar athermalizing potentials at this thickness, we decided to select the most easily obtainable and machinable material. Thus, we decided to design an athermalizing mount out of invar.

3.4) Athermalization's Results

With invar selected as our mount's material, we started to fine-tune mount's thickness to determine the dimension that would provide complete athermalization of the system. It was found that with a thickness of 2.04 mm, the invar mount creates a stress

curve that is almost constant throughout the temperature evolution, signifying that the device is successfully athermalized.

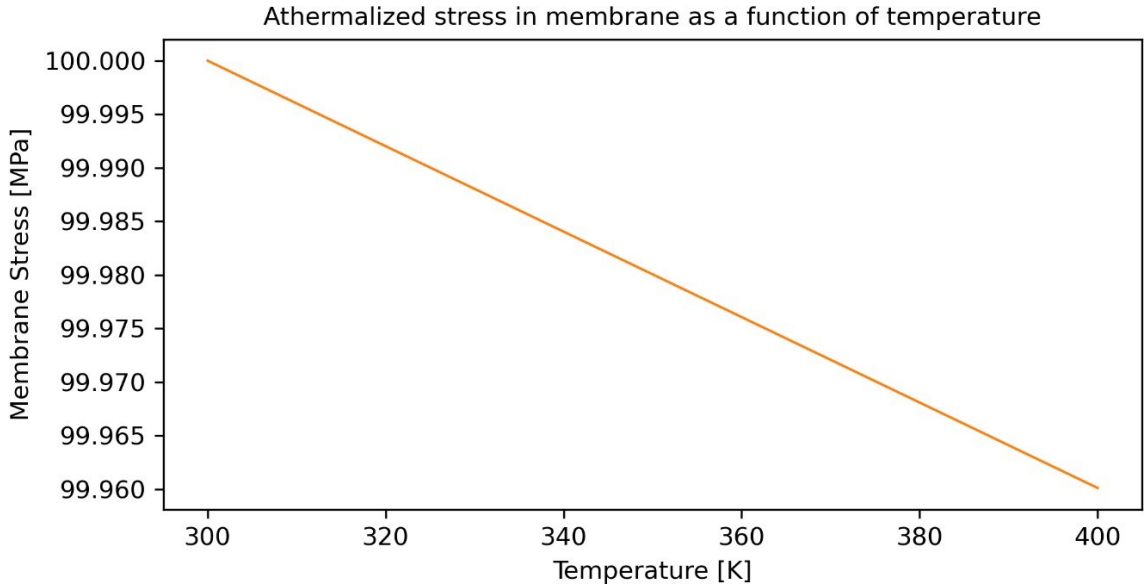


Figure 12: Stress in membrane as a function of the designed invar athermalizing mount

A table displaying the optimal thickness of each material to ensure athermalization of the system can be found below. The results are coherent with the conclusions drawn from **Figure 11** which demonstrates that, for the same thickness, Astrositall® has the best athermalization followed respectively by invar, ZERODUR® and fused quartz.

Table 2: Optimal thickness for athermalization with different mount materials

Material	Optimal Athermalizing Thickness (mm)
Invar	2.0400
ZERODUR®	2.0448
Astrositall®	1.9790
Fused Quartz	2.6460

4) Conclusion

This project aimed to find a way of neutralizing the unwanted noises coming from the temperature drift of a silicon nitride membrane and silicon substrate used for radiation sensing in our lab. The athermalization of the system would eliminate the undesired membrane's frequency fluctuation enabling us to gather more precise results in the future. Unlike the current athermalization techniques, our method of suppressing thermal drift had to be simple, compact and easy to implement. We decided to design a third layer that would be added bellow the substrate to induce bending and create a compensating stress that neutralizes the unwanted stress due the thermal expansion. The compensating stress is composed of a bending and an axial component.

After defining the various equations forming the compensating stress, a python code was created to find the mount's optimal parameters. Through the analysis of the equations and the generated plots, it was established that the main factor impacting athermalization was the thickness of the mount, while the principal influencing properties were the coefficient of thermal expansion and the modulus of elasticity. Analysis of the results revealed that there was a minimal thickness at which the mount had a negative (compensating) stress. It also revealed that the coefficient of thermal expansion had a greater impact on athermalization than the modulus of elasticity. For the former, the smaller, the better the athermalization, while for the latter, the greater, the better the athermalization. With these crucial parameters identified, we have designed mounts out of 4 suitable materials: invar, ZERODUR®, Astrositall® and fused quartz. The optimal choice to completely athermalize the system, based on commercial availability and performance, is a carefully designed 2.04mm mount made from the iron-nickel alloy, invar.

5) References

[1] T. W. Clyne, “Residual Stresses in Surface Coatings and Their Effects on InterfacialDebonding,” *KEM*, vol. 116–117, pp. 307–330, Dec. 1995, doi: 10.4028/www.scientific.net/KEM.116-117.307.

[2] D. Hälg, “Membrane-Based Scanning Force Microscopy,” *Phys. Rev. Applied*, vol. 15, no. 2, 2021, doi: 10.1103/PhysRevApplied.15.L021001.

[3] T. Larsen *et al.*, “Ultrasensitive string-based temperature sensors,” *Appl. Phys. Lett.*,vol. 98, no. 12, p. 121901, Mar. 2011, doi: 10.1063/1.3567012.

[4] N. Snell, C. Zhang, G. Mu, and R. St-Gelais, “Nanowatt Thermal Radiation Sensingusing Silicon Nitride Nanomechanical Resonators,” in *2020 Photonics North (PN)*, May 2020, pp. 1–1, doi: 10.1109/PN50013.2020.9167002.

[5] S. Schmid, L. G. Villanueva, and M. L. Roukes, *Fundamentals of NanomechanicalResonators*. Cham: Springer International Publishing, 2016.

[6] C. Zhang, M. Giroux, T. A. Nour, and R. St-Gelais, “Thermal radiation sensing using high mechanical Q-factor silicon nitride membranes,” in *2019 IEEE SENSORS*, Oct. 2019, pp. 1–4, doi: 10.1109/SENSORS43011.2019.8956551.

[7] M. Piller, N. Luhmann, M.-H. Chien, and S. Schmid, “Nanoelectromechanical infrared detector,” in *Optical Sensing, Imaging, and Photon Counting: From X-Raysto THz 2019*, Sep. 2019, vol. 11088, p. 1108802, doi: 10.1117/12.2528416.

[8] X. C. Zhang, E. B. Myers, J. E. Sader, and M. L. Roukes, “Nanomechanical Torsional Resonators for Frequency-Shift Infrared Thermal Sensing,” *Nano Lett.*,vol. 13, no. 4, pp. 1528–1534, Apr. 2013, doi: 10.1021/nl304687p.

[9] M. Wang, R. Zhang, R. Ilic, V. Aksyuk, and Y. Liu, “Frequency Stabilization of Nanomechanical Resonators Using Thermally Invariant Strain Engineering,” *NanoLett.*, vol. 20, no. 5, pp. 3050–3057, May 2020, doi: 10.1021/acs.nanolett.9b04995.

[10] H. Torun and H. Urey, “Thermal deflections in multilayer microstructures and athermalization,” *Journal of Applied Physics*, vol. 100, no. 2, p. 023527, Jul. 2006,doi: 10.1063/1.2216789.

[11] W. H. Cheng, S. F. Chi, and A. K. Chu, “Effect of thermal stresses on temperaturedependence of refractive index for Ta2O5 dielectric ®lms,” *Thin Solid Films*, p. 5,1999.

[12] M. R. Hamrour and S. Galliou, “Analysis of the infrared sensitivity of a quartz resonator application as a thermal sensor,” in *1994 Proceedings of IEEE Ultrasonics*

Symposium, Oct. 1994, vol. 1, pp. 513–516 vol.1, doi:10.1109/ULTSYM.1994.401640.

[13] R. St-Gelais, S. Bernard, C. Reinhardt, and J. C. Sankey, "Swept-Frequency Drumhead Optomechanical Resonators," *ACS Photonics*, vol. 6, no. 2, pp. 525–530,Feb. 2019, doi: 10.1021/acsphotonics.8b01519.

[14] "COMSOL: Multiphysics Software for Optimizing Designs," *COMSOL*. https://www.comsol.com/ (accessed Apr. 05, 2021).

[15] "schott-zerodur-general-may-2013-eng.pdf." Accessed: Mar. 29, 2021. [Online].
Available: https://www.schott.com/d/advanced_optics/e532f55f-d6c1-4748-8c60-886eaca1daf5/1.2/schott-zerodur-general-may-2013-eng.pdf.

[16] "Welcome to Star Instruments Online!" http://www.star-instruments.com/russian.html (accessed Mar. 29, 2021).

[17] "Technical Glass Products: Properties of Fused Quartz," *Technical Glass Products*.https://technicalglass.com/technical_properties/ (accessed Mar. 29, 2021).

[18] "GibbPresentation.pdf." Accessed: Mar. 30, 2021. [Online]. Available: https://wp.optics.arizona.edu/optomech/wp-content/uploads/sites/53/2016/10/GibbPresentation.pdf.

[19] “Crystaltechno - Aspherical Optics.”

http://www.crystaltechno.com/asph_en.htm(accessed Mar. 30, 2021).

6) Appendices

A: COMSOL FEA

We tried to confirm the results of the Python code by created a model in COMSOL FEA software. The results were inconclusive since there was no match between the python code and the FEA's result. We are not sure what causes the discrepancies between the Python and FEA results, but we do know that the FEA model contains an error since the obtained results are not logical. For the purpose of archiving the information on what we tried here is a summary.

The first step was to build the components of the model by adding three different 3 mm wide layers. As seen in **Figure 13**, the upper layer with a thickness of 100 µm is the membrane, the middle layer is substrate and has a thickness of 0.55 mm and the lower layer is the mount with a thickness of 1.8875 mm. The next step was to associate materials to each layer. To match the Python code, the properties in **Table 1** were used to attribute silicon nitride to the membrane, silicon to the substrate and invar for the mount.

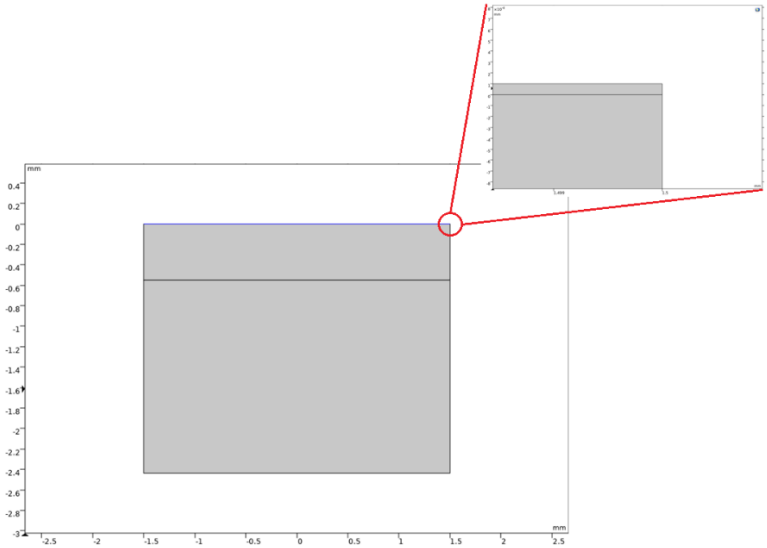


Figure 13: Components of COMSOL Model

Once the components of the model are built and attributed properties, the mesh is to be generated. The automatic generation of the mesh at its finest possible settings was selected as seen in **Figure 14**.

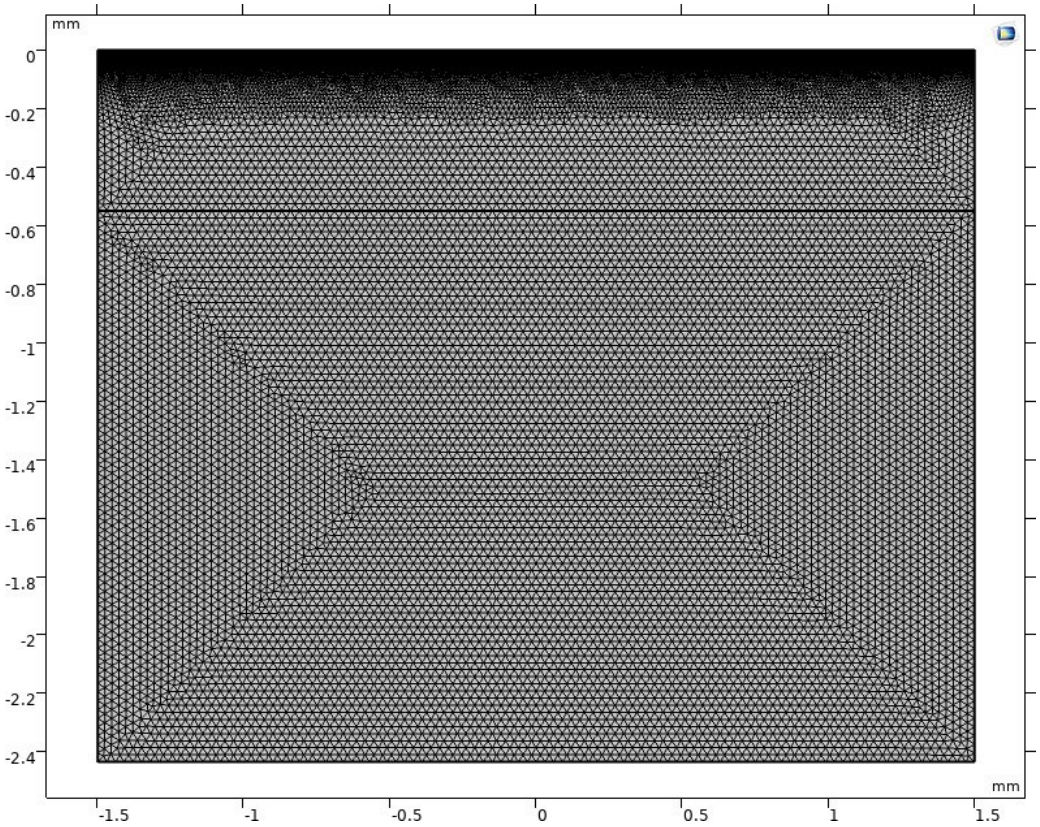


Figure 14: COMSOL Model mesh

With the mesh generated, a study must be created for the model. The study created for our case was of the stationary type and the studied physics were heat transfer in solids as well as solid mechanics. A built-in stress was applied in a stress tensor to simulate the formation stress σ_{SiN}. A sweep of temperature from 300 to 400℃ was added with increments of 1℃. A Cut Point 2D was added at the middle of the SiN membrane in order to evaluate the Von Mises stress at this location during the variation of temperature

Once the study is configured, the simulation will be executed for two different scenarios. The first scenario, as presented in **Figure 15** is one where the mount's layer is disabled in order to get data of the system without any athermalization. The second scenario has the mount layer enabled as seen in **Figure 16** allowing us to have a dataset of the athermalized system.

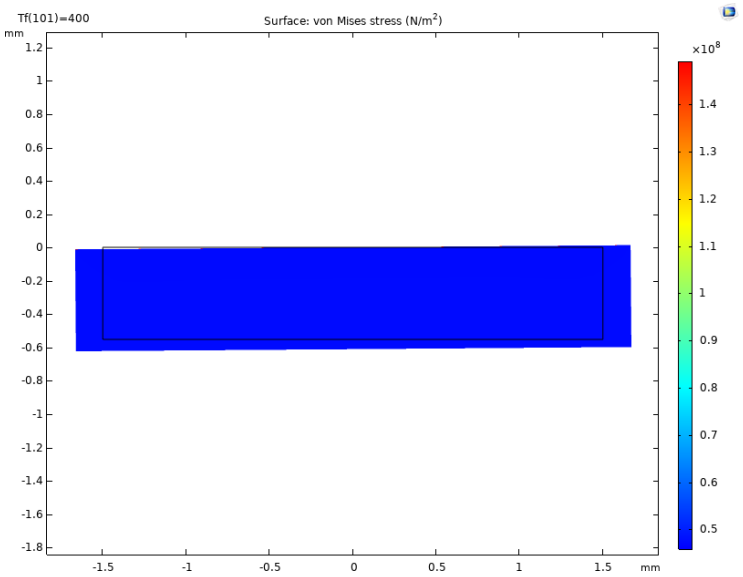


Figure 15: COMSOL Non-athermalized model simulation

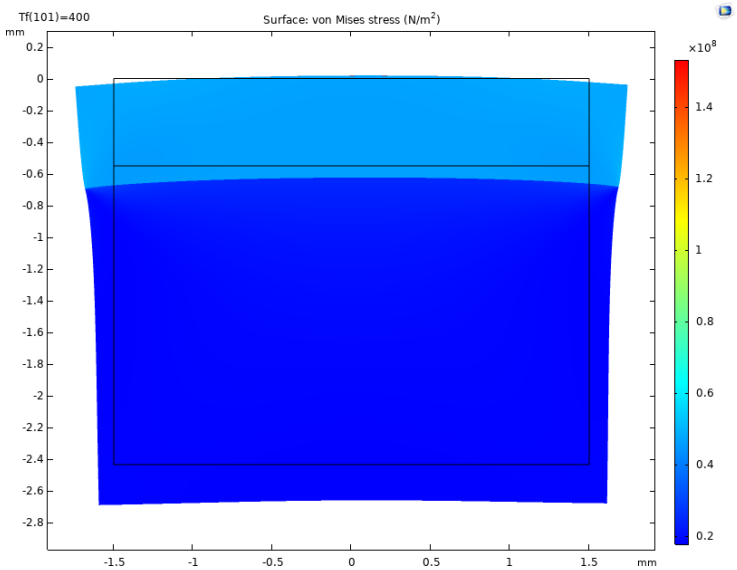


Figure 16: COMSOL Athermalized model simulation

With the simulations executed and the datasets filled, the graph comparing the COMSOL data for the athermalized and non-athermalized system with the equivalent Python model (invar mount of 1.8875 mm of thickness) can finally be done. The data is extracted to a CSV file and the results are plotted together as seen in **Figure 17**.

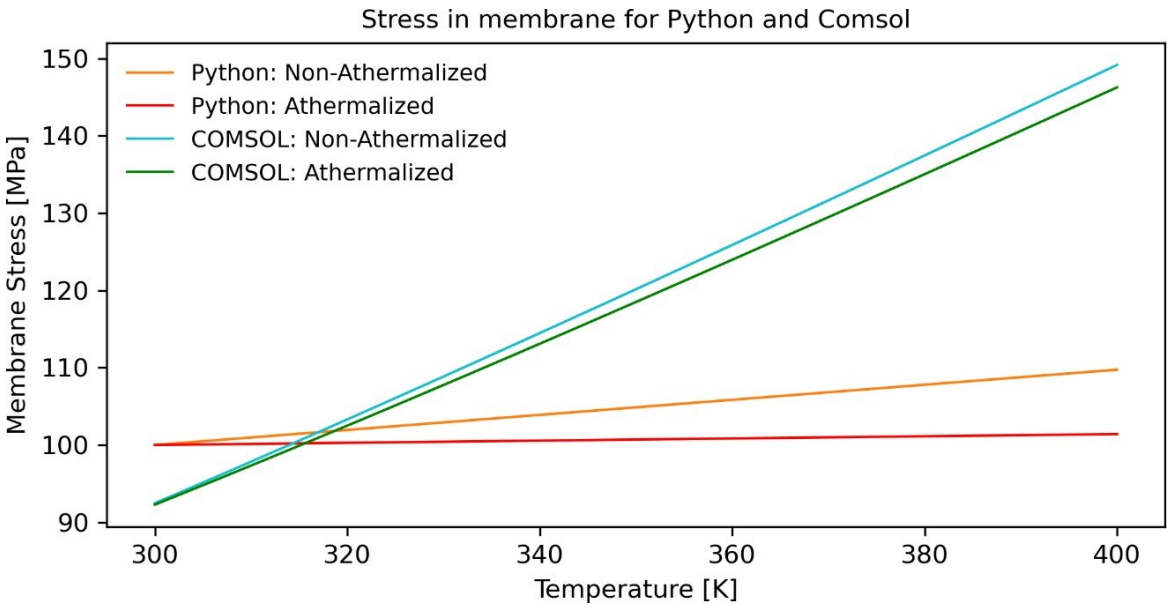


Figure 17: Comparison of results for athermalized and non-athermalized system between COMSOL FEA model and Python code

An analysis of **Figure 17** clearly depicts that the COMSOL and Python results do not match. While the difference between the non-athermalized and athermalized system for the Python model can clearly be seen, it is barely noticeable for the COMSOL model meaning that there is an error in the way its components were created, its mesh was generated or its study was configured.

An interesting observation that can be easily seen in **Figure 16** is the bending of the layered structure suggesting the model's thermal expansion is successfully working.

B: Python Code

```
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
import csv

#========== Initial Parameters ==========#
j = int(input("Enter the value corresponding to the mount's material: 1
(Invar), 2 (Zerodur), 3 (Astrositall), 4 (Fused Quartz): "))
i = j - 1
#Mount Selection
i=0
H = 2.04
#Thickness of Mount (mm)
h = 0.55
#Thickness of Silicon Substrate (mm)

Tns = np.linspace(300,400,101)
#Temperature Variation (Celcius)
T2 = 300
#Initial Temperature (Celcius

ESi = 170000
#Modulus of Elasticity of Silicon (MPa)
ESiN = 250000
#Modulus of Elasticity of Silicon Nitride (MPa)
EM = [137000, 90300, 92000, 72000]
#Modulus of Elasticity of Mount (MPa)
EM=EM[i]

CTESi = 0.0000026
#Coefficient of Thermal Expansion of Silicon (1/K)
CTESiN = 0.0000023
#Coefficient of Thermal Expansion of Silicon Nitride (1/K)
CTEM = [0.0000013, 0.00000005, 0.0000001, 0.00000055]
#Coefficient of Thermal Expansion of Mount (1/K)
CTEM = CTEM[i]

VSi = 0.28
#Poisson Ratio of Silicon
VSiN = 0.23
#Poisson Ratio of Silicon Nitride
VM = [0.29, 0.24, 0.28, 0.17]
#Poisson Ratio of Mount
VM = VM[i]

ESi = ESi/(1-VSi)
#Biaxial Modulus of Elasticity of Silicon (MPa)
ESiN = ESiN/(1-VSiN)
#Biaxial Modulus of Elasticity of Silicon Nitride (MPa)
EM = EM/(1-VM)
#Biaxial Modulus of Elasticity of Mount (MPa)
```

```
#========== DELTA CALCULATION ==========#
D = ((h*h*ESi)-(H*H*EM))/(2*(h*ESi+H*EM))
#print('The value of D is:', D, 'mm \n')

#========== AXIAL LOAD CALCULATION ==========#
DeltaE = (-Tns+T2)*(CTEM-CTESi)
#print('The value of DeltaE is:', DeltaE, 'mm \n')

#========== CURVATURE OF THE BEAM CALCULATION ==========#
K =
(6*ESi*EM*(h+H)*h*H*DeltaE)/((pow(ESi,2))*pow(h,4)+4*ESi*EM*pow(h,3)*H+6
*ESi*EM*pow(h,2)*pow(H,2)+4*ESi*EM*h*pow(H,3)+pow(EM,2)*pow(H,4))
#print('The value of K is:', K, '1/m \n')

#========== P/B ==========#
PoverB = DeltaE*((h*ESi*H*EM)/(h*ESi+H*EM))
#print('The value of PB is:', PoverB, ' \n')

#========== AXIAL STRESS OF DEPOSIT CALCULATION ==========#
SigmaD = -PoverB/h+ESi*K*(h-D)
#print('The value of SigmaD is:', SigmaD, 'MPa \n')

#========== ELONGATION OF Si ==========#
EpsilonSi = SigmaD/ESi
#print('The value of EpsilonSi is:', EpsilonSi, 'mm/mm \n')

#========== ELONGATION OF SiN Due To Mismatch ==========#
EpsilonSiN = EpsilonSi
#print('The value of EpsilonSiNM is:', EpsilonSiNM, 'mm/mm \n')

#========== TOTAL ELONGATION OF SiN  ==========#
EpsilonC = EpsilonSiN+(CTESi)*(Tns-T2)-(CTESiN)*(Tns - T2)+100/ESiN
#print('The value of EpsilonC is:', EpsilonC, 'mm/mm \n')

#========== STRESS OF SiN ==========#
SigmaT = EpsilonC*ESiN
#print('The value of SigmaT is:', SigmaT, 'MPa \n')


plt.figure(figsize=(3.3, 1.5),dpi=300)
#Graph Size

font = {'family': 'Arial', 'weight': 'normal', 'size': '10'}
#Graph Format

plt.plot(Tns,  SigmaT, '#ff7f0e' ,linewidth=1.0)
#Graph Plotted Data

plt.title("Athermalized stress in membrane as a function of
temperature", fontsize=10)          #Graph Title
plt.xlabel("Temperature [K]",fontsize=10)
#Graph X Label
plt.ylabel("Membrane Stress [MPa]",fontsize=10)
#Graph Y Label

plt.show()
#Graph Show Plot
```