Report to the American Marshall Plan Foundation

Electromagnetic probing of doped helium nanodroplets

LORENZ KRANABETTER

07. August 2018

Contents

Abstract

This report to the Austrian Marshall Plan Foundation summarizes the outcome of the short time scholarship at department of Physics and Astronomy, University of Southern California in Los Angeles. The goal was to gain a deeper understanding of helium nanodroplets and the via electron bombardment induced charge transfer processes within them. The position and size dependent deflection of a neutral doped helium nanodroplet beam is used to probe the charge transfer probability to the solvated dopant. Helium nanodroplets are produced via supersonic expansion and subsequently doped with caesium iodide or dimethyl sulfoxide. The beam is then deflected by an inhomogeneous electrical field. After ionization via electron bombardment the beam is detected with a secondary electron multiplier. For species with a known dipole moment the measured deflection profiles can be compared with a simulated profile to calculate the neutral beams size distribution. This distribution is used to calculate the parametrized charge transfer probability to compare to models found in the literature. The work conducted during the visit at Prof. Vitaly Kresins lab focused on improving the involved evaluation method and experimental setup to achieve the above-mentioned goals.

Introduction

This chapter gives an overlook over the reasoning that led to the development of the deflection setup in its current state.

Beam Deflection Experiments

One of the key experiments to observe quantum mechanical effects, the Stern-Gerlach Experiment, was designed to measure directional quantization of the angular momentum. This effect was predicted by Peter Debye and Arnold Sommerfeld while investigating the Zeeman-Effect in 1916. After the experiment was performed for the first time in 1922, before the concept of the electron spin was introduced, the results were quite different from the predicted outcome, as projected by the Bohr-Sommerfeld theory. Even though the quantization of the spatial orientation of the intrinsic magnetic momentum was validated, the correct interpretation of the results had to wait until the ½ spin of the electron came through.

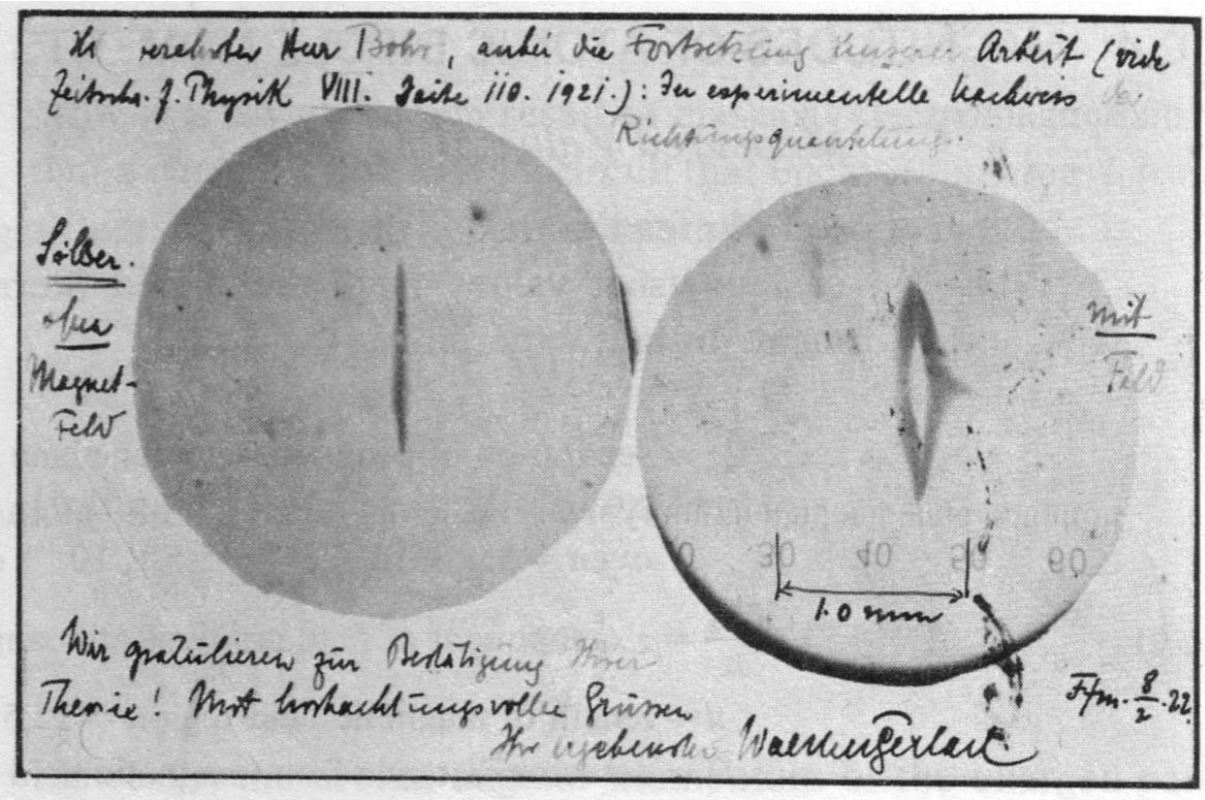

Figure 1: The results of the Stern-Gerlach experiment sent to Niels Bohr by Walther Gerlach via post card [1].

Stern designed the experiment as a beam of silver atoms that passes through an inhomogeneous magnetic field. Because of the intrinsic angular momentum, the magnetic dipole should precess in this field, much like a classical spinning dipole. The importance of the experiment was to display that this precession has certain values and is not distributed continuously up to a maximum value, very unlike a classical spinning dipole.

Besides the very important deed of convincing physicists that quantization at atomic scale is a reality, the setup makes it possible to calculate magnetic dipoles via measuring the deflection of the beam. This is also true for electrical dipoles when using an inhomogeneous electric field. This method of measuring dipole moments was adapted many times since then. Today's typical atomic or molecular beam setup is able to reach a magnetic or electric deflection in the order of fractions of mm. This

basically limits those measurements to particles with a very high dipole moment or species which can be manipulated into a highly aligned state. The latter can be accomplished by reaching ultra-cold temperatures where the statistic dealignment of dipoles via the random thermal motion is suspended. A very powerful technique to cool down particles was proposed in 1975 by two independent research groups and later demonstrated only three years later in 1978. This method is called laser cooling and was adapted successfully by many researchers to reduce the random thermal velocity in atomic beams. Because the principal of operation utilizes the energetic structure of the particles to be cooled, it is not easily applicable to complicated species. In fact, its use is only practicable up to diatomic molecules and its efficiency is highly dependent on the atom or molecule. Later developed methods that use electric-optical fields face the problem that the internal degrees of freedom are not cooled at all and even collisional cooling leaves one with a vibrational energy of a few Kelvin. Another source of cold beams is the supersonic expansion of a gaseous or liquid sample, which yields a high flux particle beam whose ultimate temperature is highly dependent on the species.

In order to circumvent the problem of species sensitive cooling efficiency and high rotational temperature, a method was adapted that could produce a wide variety of high flux highly aligned cluster complexes to enhance the achievable deflection. This matrix assisted assembly imbeds the species of choice inside helium nanodroplets, which can enhance the dipole alignment and therefore deflection. First adapted in 1990 [2] this method showed good species independent cooling properties and was widely used in spectroscopy and mass spectrometry.

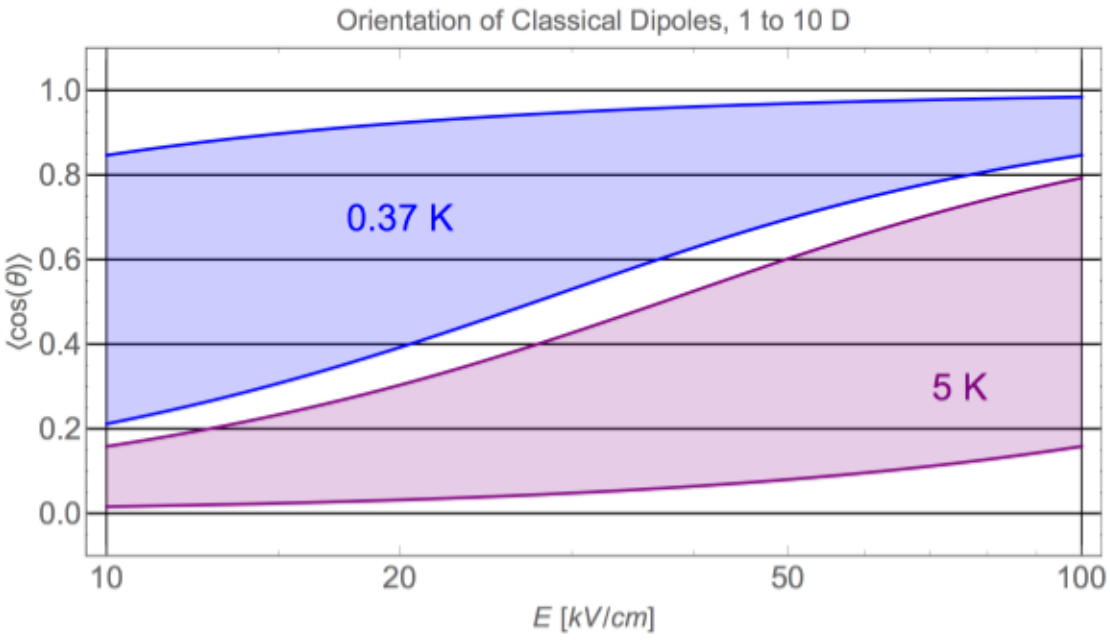

Figure 2: Orientation of dipoles in an electric field. The blue shaded area corresponds to a droplet temperature of 0.37K. The red shaded region indicates a temperature of 5K. The upper and lower boundary of the shaded areas correspond to 10 D or 1 D respectively [3].

Helium Nano Droplets

Helium nanodroplets are a unique system with properties that render them as an almost perfect matrix for experiments in almost all fields of optical and IR spectroscopy of embedded neutral and ionic analytes.

The noble gas develops most of those peculiar characteristics when cooled down far away from normal conditions. When bulk Helium reaches very cold temperatures near the absolute zero (T_b = 4.22 K), the very weak Van der Waals interaction of the atoms is strong enough to bind them in to a liquid

state. This restricts the movement of a certain atom to a confined space. Because of the low mass of the helium atom the uncertainty energy is in the same magnitude as the van der Waals interaction that confines them.

$$E_0 \sim \frac{h^2}{2m_{He}V_a^{2/3}}$$

This leads to the oppression of the solid state at pressures up to 20 bar. If cooled down even further (T_λ = 2.17 K) the liquid helium runs through a phase transition which is accompanied by some anomalies. At 2.17 K the sudden change in physical properties is easily visualized by looking at the specific heat diagram. The shape of this anomaly is the reason why this temperature value is called lambda point.

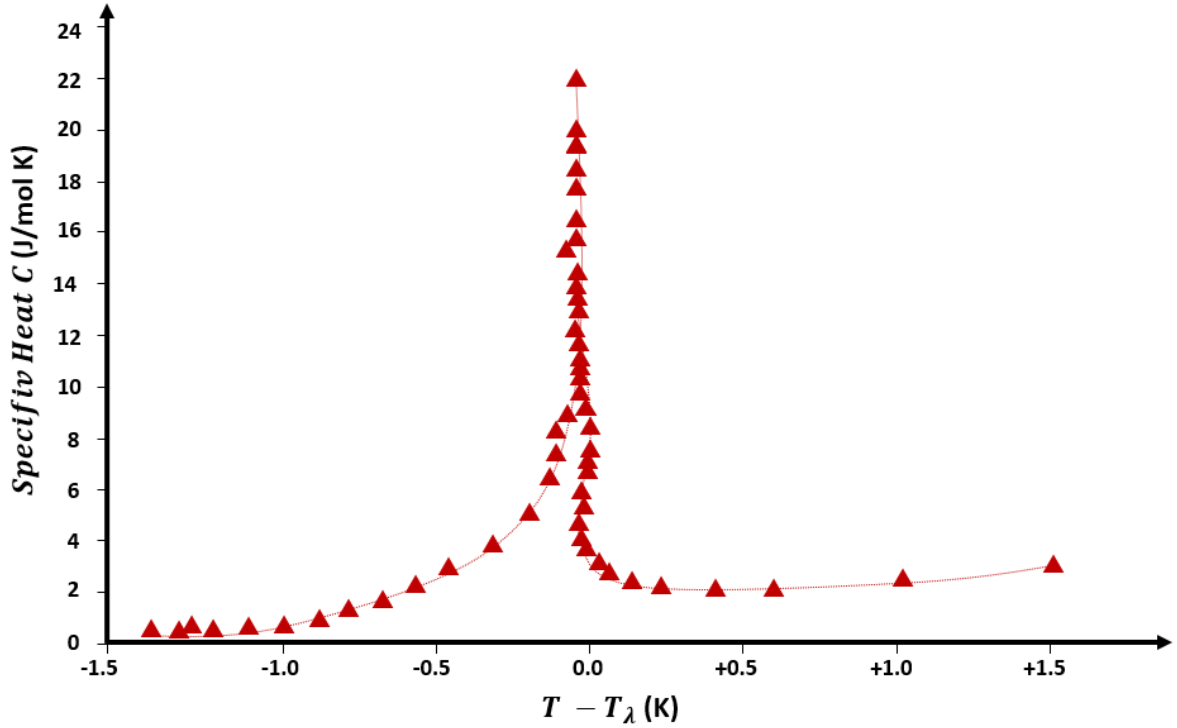

Figure 3: The lambda-transition as published by Buckingham and Fairbank in 1961 [4].

After crossing the lambda point a part of the helium is described as a superfluid [5]. The superfluid is an indivisible part of the hole fluid and lends the as He II known mixture its high thermal conductivity and almost not present viscosity. Those attributes are the reason for the good cooling properties of He II and the lack of friction when traveling at speeds below the critical landau velocity. It should be mentioned here, that superfluidity in a bulk medium does not imply that a cluster of same substance also is governed by the same formalism. In the case of He II clusters, experiments [6] have shown that helium nanodroplets from 70 atoms upwards actually show superfluid behavior. However, the transition from the bulk to the cluster phase changes some parameters of the phase shift in a not trivial way. For example, the lambda point is not a fixed temperature value but a function of cluster size.

Another remarkable attribute of the helium cluster is its self-cooling capability. A superfluid helium nanodroplet will try to keep its temperature at approximately 0.4 K. This is accomplished by the very low binding energy of a single atom to the cluster. If an atom or molecule enters a droplet it will dissipate its kinetic energy into the droplet until it reaches the landau velocity. At these speeds the

weak potential well of the droplet will be strong enough to confine the dopant on a trajectory around its center [7]. Most vibrational modes will then be cooled out while traveling inside the helium, which leaves the dopant at a very low temperature. All the energy that was transferred to the droplet will leave it via the evaporation of atoms. This sequence of events leaves an ultra-cold assembly of molecules within a matrix which engages in almost no interaction with the dopant, light and electric or magnetic fields.

This is the reason why helium nanodroplets are dubbed as the ultimate matrix for spectroscopy. The low temperature and high light permeability enables scientists to measure rotationally resolved spectra of big molecules. The same circumstances render them as a good matrix to align dipole moments within. The low temperature ensures a high initial alignment which is further enhanced by the free rotation inside the He II environment when a field is applied. While the low polarizability of the helium only impedes the electric or magnetic field in a controllable manner.

To summarize, the before mentioned properties of the helium matrix will enable dipolar molecules to align to an extraordinary degree when an external field is applied. This leads to an increase in precession induced deflection inside an inhomogeneous electric field of up to an order of magnitude.

This large increase in deflection is more than enough to compensate for the higher mass due to the attached helium atoms. But there is one disadvantage compared to conventional deflection methods that cannot be equalized by other enhanced characteristics of the helium matrix. Helium nano droplets are produced via supersonic expansion, a method that utilizes adiabatic cooling and other effects attached to the free jet expansion. Via environmental control a variety of sizes can be produced but the standard variation of the mean droplet size is always a rather broad one. Opposed to the unambiguously defined mass of the deflected particles in a classical deflected beam setup, this requires to an intensive evaluation of the deflection profiles. Because the measured deflection profiles are convoluted with the size distribution of the helium nano droplet beam, a reference particle has to be measured. The deflection of this calibration atom or molecule, which has a well-defined dipole moment, is then used to calculate the dipole moment of the species of interest from its deflection.

Even though the size distribution of the neutral helium beam prohibits a more direct measurement of the dipole moments, the advantages compared to other methods are overwhelming. Alignment with laser light for example is able to reach similar alignment quality but face different problems. The high light intensities needed to align molecules with low dipole moments may stimulate excited states. Furthermore, such intensities are only achievable with pulsed lasers systems and a laser pulse is simply limited in its shape and length which restricts the alignment duration and spatial area of effect. Another benefit of the helium nano droplet beam deflection is that it conveniently gets rid of all nonpolar particles via spatial separation. Particles with high dipole moments and low mass experience higher deflection values than particles with high mass and low dipole moments. This also separates species with the same exact mass and dipole moment, but a different amount of helium atoms attached to them. This separation allows size selection of neutral helium nano droplets and therefore the possibility to investigate the impact of droplet size on spectral analysis or ionization cross section of neutral molecules embedded in neutral helium clusters.

In order to discuss the experimental possibilities in a more accurate manner, the next paragraph is focused on the experimental methods.

Experimental Methods

A very detailed explanation of the experimental setup and the theoretical background of the experiment can be found in the PhD thesis of Danial Merthe [3].

Supersonic Expansion

A free jet expansion into the vacuum is used to produce the helium nano droplets. The pre-cooled bulk helium is pressurized and expands from a small reservoir trough a small nozzle like aperture into an ultra-high vacuum. Dependent on the chosen pressure, pre-cooling temperature and nozzle diameter, a different mean size of droplets can be accessed. Regardless of the chosen temperature, the supersonic expansion yields a beam of helium nanodroplets at approximately 0.4 K with a very sharp velocity distribution (standard deviation less than 3% of the average velocity). The mean size, mean beam velocity and flux will differentiate with the prevailing expansion properties. If the expansion starts form gaseous helium, adiabatic and Joule-Thompson cooling is lowering the temperature far enough to enable condensation of the droplets after some initial three body collisions of helium atoms. This so called sub-critical expansion regime is only able to produce small cluster sizes in the range of 10^3 to 10^4 constituents. If the pre-cooling is as efficient as to reach the “liquid” bulk phase of helium the expansion is called super-critical and the initial droplets are formed via the fragmentation of a liquid jet via turbulent flow in the separation layer. (Because of the high pressure present in the reservoir, the helium is beyond the critical point, which means that a clear differentiation between liquid and gas is not possible any more. Therefore, it is technically not correct to define the phase in the reservoir as liquid, the helium is actually “liquefying” when the pressure drops while exiting the nozzle). When the reservoir is pumped down to even lower temperatures near or below the lambda point, the helium undergoes another phase transition to enter the super liquid phase. Because of the lack of friction and turbulent flow the mechanism changes into a regime where the surface tension is the main driving force of the fragmentation. This process is then called Rayleigh-breakup and yields the biggest cluster sizes of up to 10^{12} He-atoms.

It is important to mention that all of those production methods spawn shockwaves through interaction with the residual gas of the high vacuum or the vacuum vessel geometry. These shockwaves raise the need for skimming the jet to extract a beam. The shock phenomena associated with the free jet interacting with the residual gas are called barrel-shock and Mach-disk and their dimensions depend mostly on the background pressure in the vacuum chamber. Only the latter one is of importance for the experiment because it defines the dimension of the zone of silence alongside the beam axis. The unperturbed beam needs to be extracted from this zone, which dictates the position of the skimmer element. The shockwave produced via interaction with the walls of the chamber depends almost solely on the overall flux of the jet and is the reason for the cone shape of the skimmer.

Because a considerable amount of the jet is skimmed away, the cluster source creates a heavy gas load. This combined with the high terminal compression values needed to keep the cold surfaces of the pre-cooling stage form collecting ice when no helium load is present lead to the requirement for a high-end pumping system.

After the initial droplets are created, evaporation cooling changes their temperature to the before mentioned 0.4 K.

Pickup Process

The helium nanodroplet beam exits the cluster source with speeds in between 200 m/s to 500 m/s according to the expansion conditions. If a foreign particle comes across such a droplet and the relative velocity of the droplet and the particle exceeds the critical Landau velocity necessary to induce friction in a superfluid, the kinetic energy of the particle corresponding to this velocity difference between dopant and droplet will be dissipated into the helium droplet. The dissipation continues until there is no helium left or the dopant reaches the Landau critical velocity. The particle is confined in the droplet if its kinetic energy at the critical landau velocity is lower as the dopant to droplet binding energy. The potential well inside the helium matrix is calculated to lead to particle in a box like states [8]. There are some circumstances that complicate the particle droplet interaction quite a bit (like heliophobic dopants and vorticities in large droplets), but for the droplet sizes and species in this experiment a simple representation of pick up processes is sufficient.

$$p_{k=} \frac{< k >^k}{k!} e^{-<k>}$$

Where $< k >= n6_p L$ is the average number of dopants in a droplet. With n as the dopant density per volume, 6_p as the pickup cross section and the path length trough the pickup cell L. Assuming the pickup cross section scales proportionally to the geometric cross section of the droplets. The total probability of picking up k dopants can be calculated by integrating over the cluster size distribution.

$$p_k = \int_0^{+inf} dN \; \frac{(< k >_n N^{2/3})^k}{k!} \; e^{-<k>_n N^{2/3}} \; \frac{1}{N\, 6\, \sqrt{2\pi}} \; e^{-\frac{(\ln N - \mu)^2}{2\, 6^2}}$$

Where $< k >_n$ is the average number of droplets for a number of n constituents. This integral can be approximated via different approaches, as described in Ref. [3] and [9].

All the energy dissipated into the droplet by a dopant leaves the droplet via evaporation of the fastest He atoms from the surface. This evaporative cooling is fast enough to assure that a dopant is cooled down to equilibrium temperature before it encounters another dopant. The number of atoms ejected from the droplet per dopant is in direct relation to the relative velocity to the droplet v_{diff}, the mass of the dopant m_{dop} and its internal energy $\varepsilon_{int} = \varepsilon_{rot} + \varepsilon_{vib} + \varepsilon_{bin}$.

$$\Delta N = \frac{m_{dop}\, {v_{diff}}^2 + 2\, \varepsilon_{int}}{2\, D}$$

Because of the huge mass difference between the dopant and the helium nano droplet the change in velocity due to the pickup process is negligible. In addition, solvation energy of the dopant into the surrounding He matrix as well as binding energy in case of dopant cluster formation leads to a heat transfer to the droplet and subsequent He evaporation.

Beam Deflection

After the pickup chambers, the helium nanodroplet travels towards the deflection stage. Rotational spectroscopy has shown, that a helium nano matrix modifies the rotational constant of molecules. A modern theoretical approach to this phenomenon is the introduction of a quasiparticle called angulon. Those describe basically a quantum rotor dressed with many particle field excitations of the surrounding helium matrix [10]. In general, this theory describes the helium matrix by a reduction in the effective rotational constant, especially if the helium-molecule potential is highly anisotropic. Which raises the susceptibility to external fields therefore increasing the average alignment of a thermalized sample to the field orientation.

In order to get a net force on a dipole, an inhomogeneous electrical field is required. This field is shaped by a Rabi two wire setup. Because of the shape and small deflection within the electric field, the change in alignment to the field orientation and therefore the change in effective dipole moment of the sample can be assumed to be negligible. The equation of motion, considering there is no field gradient in the y and z axis is

$$m\,\ddot{x} = p_{eff}\frac{\partial \epsilon}{\partial x} \sim p_{eff}\left[\left(\frac{\partial \epsilon}{\partial x}\right)_{x=0} + x\left(\frac{\partial^2 \epsilon}{\partial x^2}\right)_{x=0}\right]$$

Where p_{eff} is the effective dipole moment of the whole complex consisting of the helium droplet with the induced dipole p_{He}, and the dipole of the dopant $p\langle\cos\theta\rangle_{thermal}$.

$$\langle\cos\theta\rangle_{thermal} = -\frac{1}{p}\frac{\partial}{\partial \epsilon}\ln\sum_E e^{-E/k_bT}$$

p_{He} can be calculated by treating the droplet as a classical dielectric sphere using the permittivity of bulk He II at 0.4 K. Solving these equations suggests that even doped droplets with 10^4 atoms can reach measurable deflection values [3].

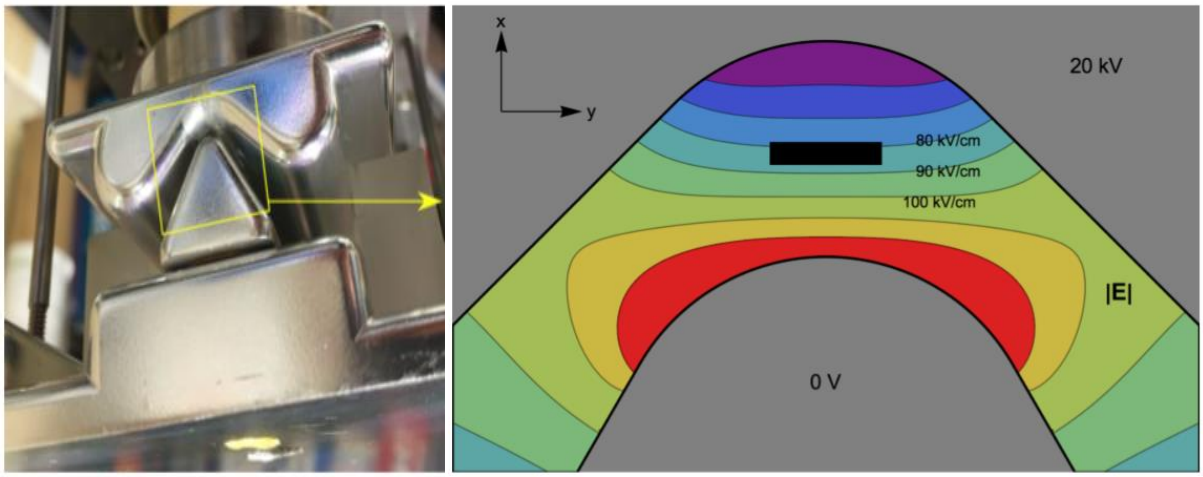

Figure 4: The strongly inhomogeneous electrical fields are generated via the depicted aluminum profiles. The cross section of the electrodes shows the shape of the electrical field. The beam position is indicated by the black rectangle [3].

Ionization

Instead of a simple glass plate, as used in the first deflected beam experiments, this setup uses a quadrupole mass filter with a subsequent electron multiplier for the detection of individual particles in the beam. The detection unit needs the neutral beam to be ionized for detection, which is accomplished via secondary processes induced via electron bombardment of the beam.

After an aperture blocks all particles except those which are deflected to a certain solid angle, the helium droplet beam is crossed with an electron beam. When an incident electron encounters a helium nano droplet, it uses up some kinetic energy to enter the helium environment. Dependent on the energy it has left, the electron may trigger different reaction sequences. Because of the scope of this report, only electrons with enough energy to trigger an ionization event that yields to a positively charged ion, are discussed here. This only includes electrons that are able to lose at least 19.8 eV of kinetic energy to inelastic scatter while still be able to escape the atomic potential of the collision partner.

Electrons with enough energy can transverse the droplet quiet easily to interact with multiple helium atoms. The most prominent events for such electrons are ionization (24.6 eV) or excitation (19.8 eV) of a helium atom via inelastic scattering. If the kinetic energy of an incident electron is depleted before it leaves the droplet, the Pauli repulsion induced by the 1s electrons of the helium environment is creating a bubble with the electron in its center. This electron bubble seeks to surface and eject itself from the droplet. Such low energy electrons can also attach them self to exited helium to form a helium anion.

The total helium-electron scattering cross section for an electron energy of 70 eV is about $1.4\ \dot{A}^2$ while the total ionization cross section is calculated to be $0.3\ \dot{A}^2$ [11]. This combined with the number density of helium nano droplets with $0.022\ \dot{A}^{-3}$ [12] calculates to a mean free path length of approximately $33\ \dot{A}$ for scattering events. When comparing the mean free path with the possible pathway length through a droplet with about 10000 He-atoms (about $150\ \dot{A}$), the average number of scattering events can be estimated to be small. The smaller ionization cross section leaves the mean free path between ionization events at approximately $153\ \dot{A}$. This indicates that in small cluster the ionization probability of a constituent should be uniformly distributed, and the whole cluster is most likely to only carry one charge. It is important to note, that this changes with increasing cluster size.

Even if an impurity inside a cluster has a relatively large ionization cross section, the mere number of He-atoms renders the direct ionization of a dopant very unlikely. The primary ionization channels for dopants is charge transfer from a helium cation and penning ionization via an exited He-atom. Where the latter one is dominant at energies below 24.6 eV but is less important at higher energies. The general idea behind the charge hopping mechanism is that an electron hole created on a helium atom is not localized. On the timescale of an electron scattering or exchange process, the atomic motion is very slow. Even though, the charge induces an atomic motion through weak induced-dipole interaction with its neutral neighbors, the atoms need some time to react to those forces. In this timeframe, the electron hole sees a uniformly distributed assemble of virtually undistinguishable particles. Hopping from one to another has no effect to the energy of the droplet. This leads to the charge performing a random walk. As soon as the atomic motion catches up with the electron, it sees the potential well created by the closer distance between its cation and a helium atom, therefore producing He_2^+. If the hole encounters an impurity on its path the charge is going to stick with the dopant prematurely ending the delocalization of the electron hole. Both charge localization processes release energy in the surrounding helium, which leads to further evaporation of atoms. Dopants can be even stripped of all their attached helium leaving them as bear ions. Because of the offset of evaporative cooling with the departure of the last He atom, the temperature of those ions can be far off the equilibrium temperature of a still solvated ion.

There are several theories about the charge transfer in helium nanodroplets. All of them are based on the above mentioned random walk formalism but introduce a bias in the direction of the random walk via different mechanisms [13] [14] [15] [16].

Mass selection

After the ionization process, the now ionized beam of a certain deflection value is analyzed via a quadrupole mass filter. This is accomplished by orthogonally pushing low-mass fragments of it into the filter.

A quadrupole mass spectrometer utilizes the inertia of particles to select one certain charge to mass ratio to pass the filter unperturbed. To generate an ideal quadrupole four parallel rods with a hyperbolical profile are arranged in each corner of a square with an enclosing circle of radius of r_0. Rods joint together by the diagonal of the square are connected and form a pair. A time dependent electrical voltage is introduced between both pairs, consisting out of a time independent voltage U and a radio frequency voltage $V_{RF} = V\ \cos(\omega t)$. If Ions are injected axially at the center of the enclosing circle they move through a potential field ϕ.

$$\phi(x, y, z, t) = (U + V\ \cos(\omega t)) \cdot \frac{x^2 - y^2}{r_0^2}$$

Therefore, the equations of motion for a singly charged particle are

$$m\,\ddot{x}\ + 2\,e\,(U + V\ \cos(\omega t)) \cdot \frac{x(t)}{r_0^2} = 0$$

$$m\,\ddot{y}\ + 2\,e\,(U + V\ \cos(\omega t)) \cdot \frac{y(t)}{r_0^2} = 0$$

$$m\,\ddot{z} = 0$$

Using the following identities enables the equations to be written as a system of standard Mathieu differential equations.

$$\underbrace{\kappa := \frac{\omega t}{2}; \quad \alpha := \frac{8\,e\,U}{m\,r_0^2\omega^2}; \quad \beta := \frac{4\,e\,V}{m\,r_0^2\omega^2}}$$

$$\ddot{x} + (\alpha + 2\,\beta\ \cos(2\kappa)) \cdot x = 0$$

$$\ddot{y} + (\alpha + 2\,\beta\ \cos(2\kappa)) \cdot y = 0$$

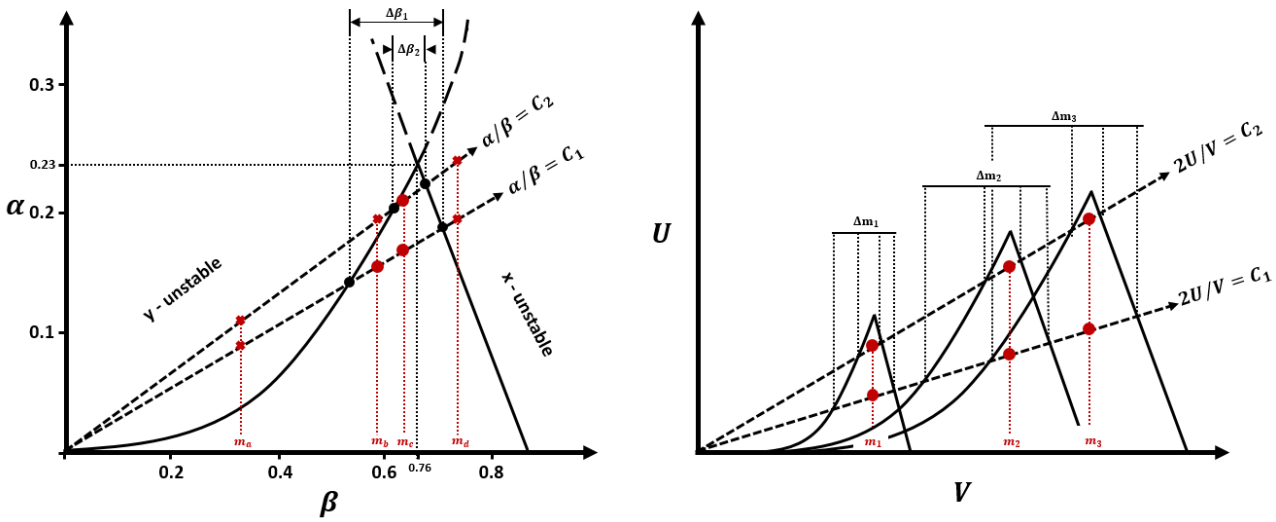

Figure 5: Quadrupole stability curves calculated via the Mathieu differential equations. The resolution of the stable mass to charge ratio is selected via the constant U/V ratio [18].

Which are well studied special forms of the Hill differential equation. The theorem of Floquet proposes that at least two solutions exist and can be found. Certain field parameters only allow ions in a certain mass range Δm to have a stable trajectory through the filter. A trajectory is stable if the maximal induced oscillation r_{max} is smaller than r_0. The ratio between U and V determines the mass resolution Δm and scanning V while keeping U/V constant conducts a mass scan with the chosen Δm. In contrast to other mass spectrometer systems, the quadrupole has a linear mass scale.

$$U_{stable} = k_u\, m_{ion}\, r_0^2\, f; \qquad k_u \coloneqq 1.2122 \cdot 10^{-8}[kg/A\,s]$$
$$V_{stable} = k_v\, m_{ion}\, r_0^2\, f; \qquad k_v \coloneqq 7.2226 \cdot 10^{-8}[kg/A\,s]$$

It is also important to understand that a U/V value smaller than 0.1678 are setting the quadrupole to an impermeable condition. While $U = 0$ sets it to a high pass setting where all particles with an $\alpha < 0.905$ are transmitted.

$$m_{ion} > \frac{k_H\, V}{r_0^2 f^2}; \qquad k_H \coloneqq 1.0801 \cdot 10^7\ [A\,s/kg]$$

Because of practical reasons the ideal quadrupole field is normally approximated via the use four rods with cylindrical profiles. If the dimensions are chosen properly this only yields neglectable differences in the near field of the rods.

When measuring deflections, the quadrupole mass analyzer is set to the desired mass to charge ratio and set to an appropriate mass resolution (the chosen $m/\Delta m$ values are highly dependent on the achievable signal strength). Then the whole detection chamber is shifted alongside the deflection axis to measure the deflection of particles that are transmitted through the mass filter. The ion flux of the mass analyzer is measured via a secondary electron multiplier. This method allows to measure mass spectra at a different deflection value or the deflection profile of certain mass to charge ratios.

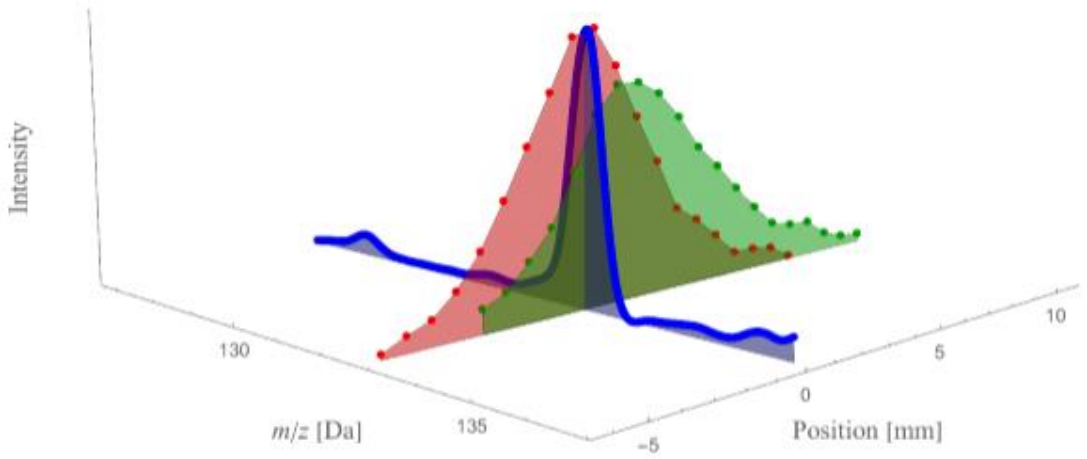

Figure 6: Visualization of measurement capabilities of the setup. The blue curve represents the mass spectra while the red and green curves depict two beam profiles at different mass to charge values [2].

In order to improve the poor signal to noise ratio achievable by evaluating the raw, amplified ion current from the secondary electron multiplier, a dual-phase digital lock in amplifier in combination with a beam chopper is used. This technique filers noise that is virtually indistinguishable from the signal in the time domain via separating signal and noise in the frequency realm.

The beam chopper turns the continuous beam into a pulsed source, with a square wave output at about 500 Hz. This modulated input is overlapped by the relatively constant background signal. The lock in amplifier takes the modulation reference signal (carrier wave) measured of the beam chopper via an optical switch and multiplies it with the input signal U_{in} from the detector. This product filtered via a low pass and integrated over a time interval T that is much longer than the period of the carrier wave gives the cross correlation of the input signal and the modulation reference signal at a certain phase shift. Therefore, it displays the difference between the carrier wave, which carries no information, and the signal. This leads to a strong attenuation of other frequencies and a dampening of matching but out of phase frequencies. This phase sensitivity enables the amplifier to measure phase shift ϕ, which can be used to calculate the beam velocity. Dual-phase lock in amplifiers measure the cross correlation towards the carrier wave twice. Once in phase with the reference frequency and a second time with a $\pi/2$ phase shift. Those two outputs are called "in-phase" (X_{out}) and "quadrature" (Y_{out}) and represent the correlation as vector relative to the modulation reference.

For simplicity the following demonstrative calculation is done with a sinusoidal carrier wave.

$$X_{out}(t) = \frac{1}{T}\int_{t-T}^{t} ds\ \sin[2\pi f_{carrier} s + \phi] U_{in}(s)$$

$$Y_{out}(t) = \frac{1}{T}\int_{t-T}^{t} ds\ \cos[2\pi f_{carrier} s + \phi] U_{in}(s)$$

Via the change into polar coordinates the need to choose a certain phase is eliminated, while keeping the phase accessible.

$$R = \sqrt{X_{out}^2 + Y_{out}^2} = V_{sig}, \qquad \phi = \arctan\left(\frac{Y_{out}}{X_{out}}\right)$$

Where V_{sig} is the amplitude at the carrier wave frequency and ϕ is the phase shift to the signal wave. Therefore, the phase difference $\Delta\phi$ for two different chopper frequencies can easily be measured. If two frequencies f_1, f_2 with a total phase difference $\Delta\phi$ of $2\,\pi$ are found, the beam velocity is calculated as follows.

$$\phi = \frac{2\pi f l}{v} \quad \longrightarrow \quad \overbrace{\phi_1 - \phi_2}^{\Delta\phi} = \frac{2\pi\, l\, \overbrace{(f_1 - f_2)}^{\Delta f}}{v} \quad \xrightarrow{\Delta\phi = 2\pi} \quad v = \Delta f \cdot l$$

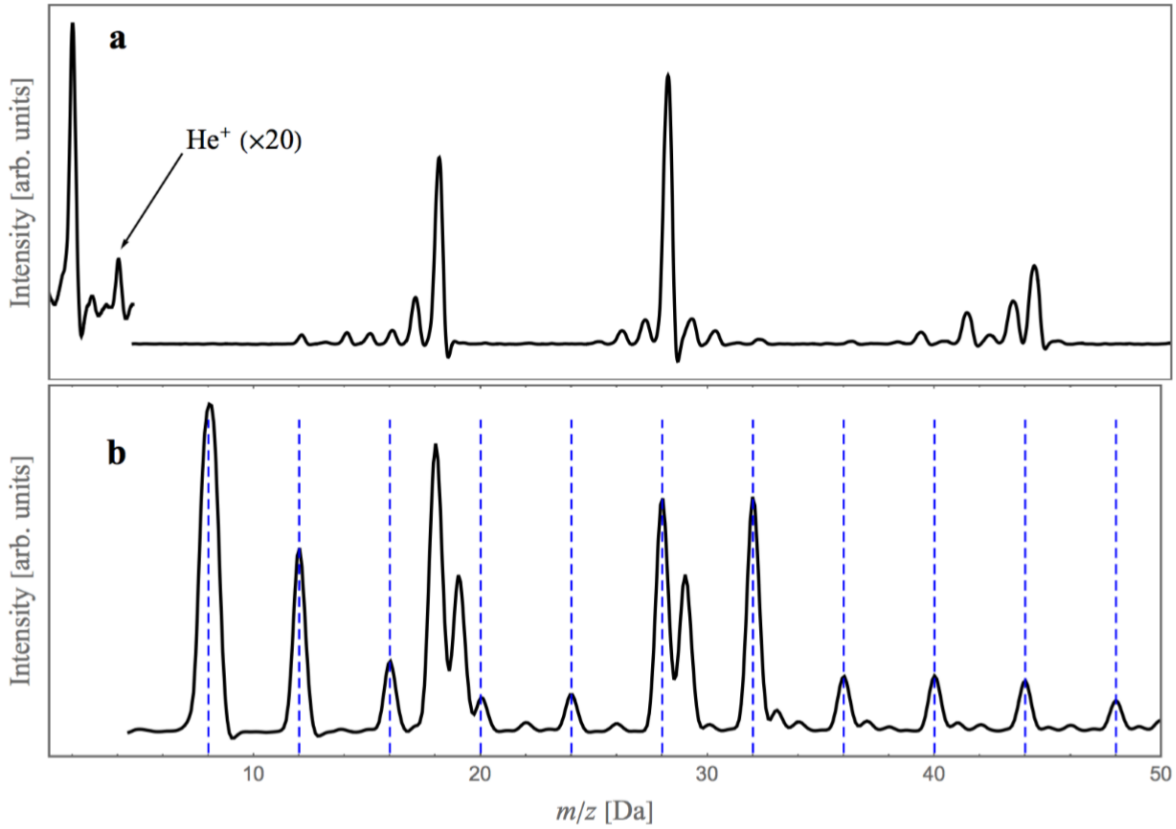

Figure 7: Comparison between a spectrum measured without (a) and with (b) a digital dual-phase lock in amplifier active [2].

Experiment and Application

Experimental Setup

The helium droplet source was built in the late 1990 and was used in combination with the still used quadrupole mass analyser to record size distributions of doped helium nanodroplets. After being decommissioned in 2007 the apparatus was restored and used to measure cold chemical reactions inside helium nanodroplets [17] in 2014. Shortly after the machine was overhauled to accommodate a deflection stage realized via two electrodes in a two-wire setup. The helium droplets are produced via supersonic expansion into a vacuum. The cold temperature of down to $10\ K$ is achieved via a two-stage cold head attached to a compressor (ARS DE-204-FF refrigerator unit) utilizing the Gifford-McMahon cycle to displace helium with a static pressure of 16 bar. Temperatures up to 30 K above the ultimate cooling power of the cryostat can be reached via resistively heating the second stage of the cold head. The heating power and therefore the temperature set point is controlled via a PID controller connected to a silicon diode on the cold head. Helium is supplied from a pressurized gas cylinder with a regulated output pressure of about $80\ bar$. The chambers are evacuated by a diffusion pump connected to a pre-vacuum stage (Chamber pressure is about $10^{-5} torr$).

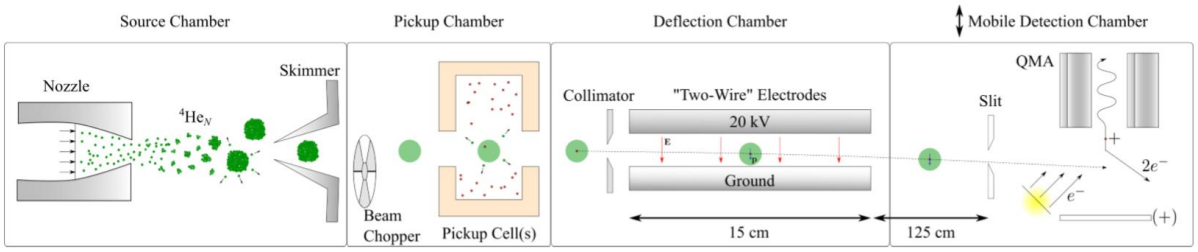

Figure 8: Schematic sketch of the experimental setup [2].

The pickup chamber includes the beam chopper and up to two cells. The pickup cells can be resistively heated and allow vapor to be injection into them from a pre-vacuum stage. The heating is controlled via PID controllers coupled with a thermocouple wire. The vacuum is pumped via a diffusion pump backed by a pre-vacuum stage (About $10^{-7} torr$).

A transferable collimator is mounted in front of the deflection electrodes, this allows beam shaping when necessary while circumventing low transmission when optimizing. The electrodes are followed by a field free region with a slit mounted to a linear manipulator at its end. The whole setup is mounted on a transversal stage form the electrode onward. The joint is realized via a below tube. After the slit the beam is ionized via electron bombardment and mass selected via a cross beam mounted quadrupole mass spectrometer (Balzers QMG 511). The vacuum is produced via turbo-molecular pumps backed by a pre-vacuum (About $10^{-8}\ torr$).

The vacuum system uses liquid nitrogen traps on critical positions to enhance the pressure (stop backflow of oil etc.) and to trap water. The pre-vacuum is produced via oil sealed rotary vane pumps.

Parametrization of Charge Transfer Models

As mentioned in [13] [14] [15] [16] there are multiple theories that give the random walk conducted by an electron hole inside the helium nanodroplet a preferable direction. Those theories predict different scaling laws for the charge transfer.

The first model introduced by Janda and co-workers [14] [15] models the charge transfer via an electron hole manifesting at a random location. The charge hopping is then modelled as a random walk with a slight bias towards the randomly positioned dopant. This bias is explained by long range monopole-dipole or monopole-induced dipole interaction. This administers the charge transfer probability to be proportional to the probability density of the dopant molecule. Because of the uniform nature of the random droplet distribution in the droplet, the charge transfer probability scales as follows.

$$P_J(x, He_n^+M \rightarrow M^+) \sim N^{-1}$$

The model of Ellis and co-workers [16] assumes that the charge is created at or in near vicinity of the droplets surface and then conducts a random walk biased towards the centre of the droplet. The survival probability after every jump is model as an exponential decay. The charge transfer is successful if the charge is able to enter the proximity of a dopant before it reacts into a helium dimer. The resulting scaling law takes the following form with the unknown coefficient g.

$$P_E(x, He_n^+M \rightarrow M^+) \sim e^{-gN^{1/3}}$$

The model of Miller and co-workers [13] yields similar scaling laws with the expectation that it does not approaches 1 when N goes to zero. Here the charge is spawned randomly in the droplet volume to do a random walk biased alongside the field lines towards the negative end of the dipolar molecule while the number of steps is also treated as a geometric series. If the charge has to leave the droplet or is too far to the positive side of the dipole the charge is not going to be transferred. This leads to an excluded volume inside the droplet from which charge transfer is impossible.

There is also the possibility of a classical unbiased random walk where the step number is drawn from a geometric distribution. This will either yield a probability equivalent to N^{-1} or $e^{-r_{drop}}$, dependent of the initial charge position. With the charge spawning randomly inside the volume of the droplet versus a random charge position on the surface.

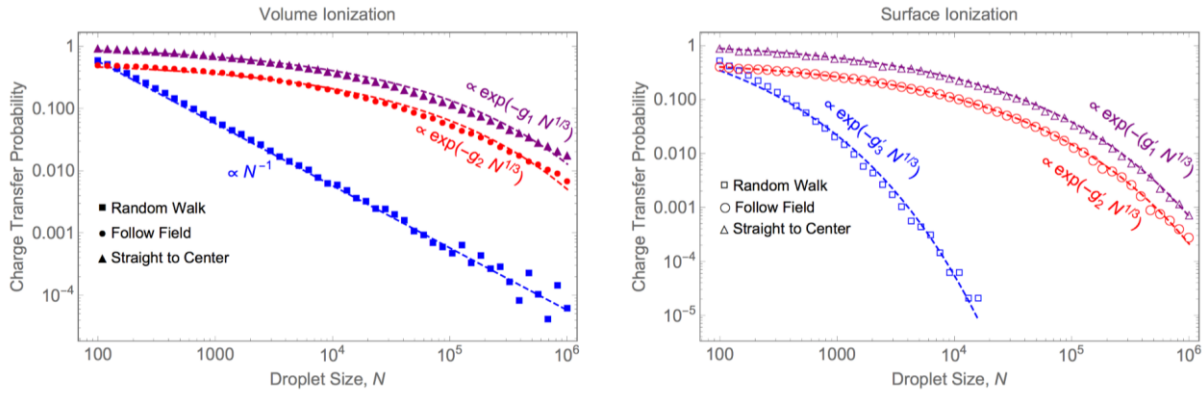

Figure 9: Charge transfer Probability according to different models [2].

Determination of the Charge Transfer Probability

Two very different molecular species where measured to characterize the charge transfer inside helium nanodroplets. Caesium iodide (CsI) and dimethyl sulfoxide (DMSO) where both picked up in via evaporation in two independent measurement campaigns. The vapor was created inside a pressure cell which can be overlapped with the helium beam trajectory. While the low vapor pressure of the liquid DSMO enables the evaporation at room temperature the solid CsI must be tightly packed into an oven and heated to reach sufficient and stable vapor pressure values for evaporation. To reach dense packing, the CsI powder is solvated in methanol and packed onto the walls of the oven. The methanol is evaporating rapidly when the camber is pumped down to operating pressure. This leaves a sufficiently dens crystalized layer of CsI conformed to the oven walls.

A measurement campaign consists out of four measurement series. First the vapor pressure curve of the dopant is analysed via the measurement of the heat or pressure dependent relative ion yield of the parent molecule or one of its fragments. This series exists of several mass spectra which are fitted with a python script utilizing the nonlinear least squares minimization and curve fitting (LMFIT; https://lmfit.github.io/lmfit-py/) package for Python. The basic method of fitting is a least minimum square algorithm using the Levenberg-Marquardt method.

The given data set is an m dimensional vector X of points and an $n+1$ dimensional model function f.

$$X \coloneqq (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$$

$$\xi = (\xi_1, \xi_2, \dots, \xi_n)$$

$$f: \mathbb{R}^n \to \mathbb{R}^m; \;\; y \coloneqq f(x, \xi), \;\; m > n$$

The desired solution is a set of parameters ξ that minimizes the sum of the squared residuals S.

$$S \coloneqq \sum_{i=1}^{m} \left(\frac{r_i}{y_i - f(x_i, \xi)} \right)^2$$

$$\frac{\partial S}{\partial \xi_j} = 2 \sum_i r_i \frac{\partial r_i}{\partial \xi_i} = 0$$

Because of the nonlinearity of the model function, the derivatives are functions of x and ξ. But after an initial parameter guess is given, the values can be refined iteratively by a successive approximation.

$$\xi_j^{k+1} = \xi_j^k + \Delta\xi_j \quad \xrightarrow{\xi_j \approx \xi_j^{k+1}} \quad f(x_i,\xi) \approx f\left(x_i,\xi^k\right) + \sum_j \underbrace{\frac{\partial f\left(x_i,\xi^k\right)}{\partial \xi_j}\overbrace{\left(\xi_j - \xi_j^k\right)}^{\Delta\xi_j}}_{J_{ij}\Delta\xi_j}$$

$$\Delta y_i = y_i - f(x_i,\xi^k)$$

$$r_i = y_i - f(x_i,\xi) = \left(y_i - f\left(x_i,\xi^k\right)\right) + \left(f\left(x_i,\xi^k\right) - f(x_i,\xi)\right) \approx \Delta y_i - \sum_{l=1}^{n} J_{il}\Delta\xi_l$$

Substituting terms in the gradient equation of S with the first order approximations given above yields the following equation

$$\frac{\partial S}{\partial \xi_j} \approx \underbrace{-2\sum_{i=1}^{m} J_{ij}\left(\Delta y_i - \sum_{l=1}^{n} J_{il}\Delta\xi_l\right) = 0}_{\sum_{i=1}^{m}\sum_{l=1}^{n} J_{ij}J_{il}\,\Delta\xi_l = \sum_{i=1}^{m} J_{ij}\Delta y_i}$$

$$\boldsymbol{(J^T J)\Delta\hat{\xi} = J^T \Delta\hat{y}}$$

The Levenberg and Marquardt introduced a dampening factor λ into this equation to optimize the individual iteration step by adjusting the dampening according to the reduction quality of S.

$$\boldsymbol{(J^T J + \lambda\, diag[J^T J])\Delta\hat{\xi} = J^T \Delta\hat{y}}$$

The advantages compared to other packages (especially the well-known *scipy.optimize.leastsq* method from which this method is extended from) is the implementation of parameter objects instead of a simple float format. This enables the *lmfit.minimize* method to fix or vary values during the fit and set upper and lower limits. Even algebraic expressions can be used to constrain parameters. The script was developed by the applicant and can be found in Appendix A.

After converting a measured spectrum from a csv file format (comma separated values) to a NumPy array, the data points are first scanned for local maxima and afterwards a guessing function is calculated via the position and number of maxima as well as the 60% intensity drop considering local minima. A on basis of the guess function individually constructed fitting function method is then assigned to the data array. This function is a linear combination of gaussian peak functions, where the gaussian shape can be convoluted with a spline function to form a custom peak shape. The parameters of the guess and the fitting function are the input for the lmfit.minimize method.

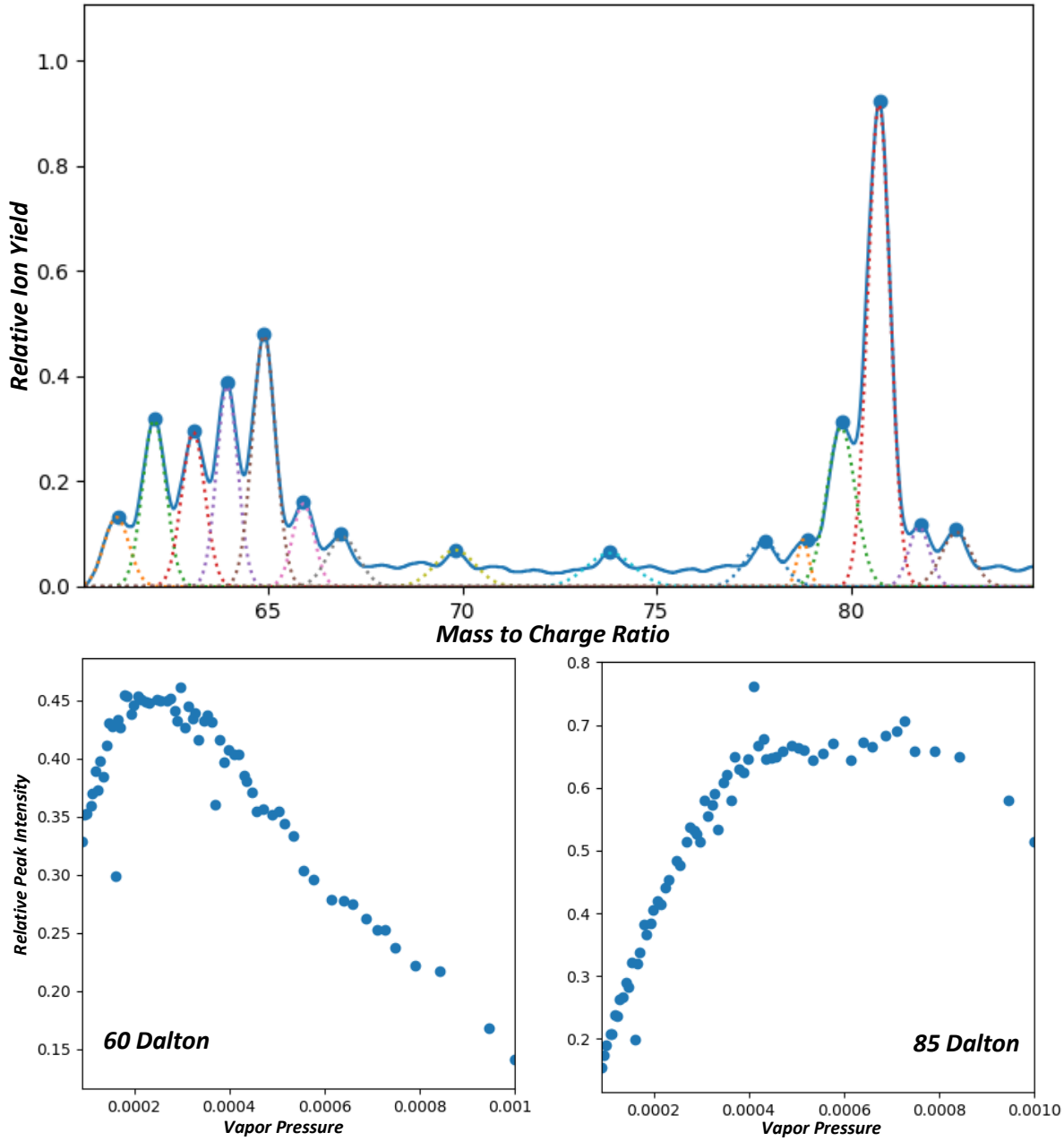

Figure 10: Fitted mass spectrum of DMSO fragments from mass to charge ratio 60 Dalton to 85 Dalton. The bottom left graph shows the area of the gauss peak function associated with the peak at 65 Dalton. The bottom right graph depicts the same for mass to charge ratio of 81 Dalton.

The top subplot in figure 10 shows the outcome of a fit with automatic parameter borders. The algorithm allows for manual adjustment of virtually any parameter or start value, which can enhance the result. Because of the heavily overlapping peak borders the signal intensity is not calculated from the raw data but from individual peaks of the fitted signal. The bottom left and right subplots show the area calculated via the trapezoid integral formulism for peaks positioned at the mass to charge values 65 and 82 respectively. The trapezoidal rule states that the definitive integral can be numerical approximated by

$$\int_a^b dx\, f(x) \approx \frac{\Delta x}{2} \sum_{n=1}^{N} \frac{f(x_{k-1}) + f(x_k)}{2}\, \Delta x_k.$$

This method is the first choice, because it is also able to calculate areas limited by a custom peak shape or even raw signal. The accuracy of his approximation leads to errors almost one magnitude lower than critical errors made through experimental necessity.

The second measurement series is a deflection analysis of the doped helium beam. The mass selector is set to a mass to charge ratio associated with the dopant, while the slid position is changed via a randomized sequence. Such a beam profile is measured for a deflection voltage of zero volt and for a deflection voltage of 20 kV.

The next measurement follows the exact same procedure with the mass selector set to transmit helium dimer ions.

The last series repeats the helium dimer deflection just with no dopants present in the experiment.

The first measurement is used to find the right pickup conditions to maximize the pickup of monomers. (Maybe something about the linear regime of the vapor pressure)

The latter three measurements are conducted to determine the charge transfer mechanism inside the helium nano droplet or to measure the dipole of a large molecule or cluster. Both measurements are in need of a molecule with a known dipole to be used as a calibration particle to parametrize the size distribution of the neutral helium nanodroplets.

The dipole moment of CsI and DMSO are well known which allows them to calibrate the beam parameters on their own.

The derived position sensitive intensity $S^{D}_{M^+}(x)$ in the deflection of the doped helium beam is solely due to helium droplets containing a dopant. It can be modelled via the following equation

$$S^{D}(x) = A \cdot \eta[I(x)^{U} + I(x)^{D} P(x)_{He}]$$

Where $I(x)$ is the position-dependent flux of droplets. The superscripted ^{U} indicates undoped droplets while the superscripted ^{D} indicate a doped inside the droplets. $P(x)_{He}$ is the charge transfer probability form the helium to the dopant and η is a normalization factor to account for scattering and collision events. A is introduced to relay the different detection efficiency for separated ion channels.

$$S^{U}(x) = A\,[I(x)^{U} + I(x)^{\not{D}}\, P(x)_{He}]$$

The above equation represents the signal of an undoped beam where the superscript $^{\not{D}}$ indicates droplets doped with an impurity out of the residual gas. $S(x)^{D}_{M}$ is the signal from the dopant.

$$S(x)^{D}_{M} = B\,\eta[I(x)^{D}_{M} P(x)_{M}]$$

$$I(x)^D = I(x)_M^D + I(x)^{\cancel{D}}$$

The above substitution incorporated into the equation yield

$$S(x)^U = A\,[I(x)^U + I(x)^{\cancel{D}}\,P(x)_{He}]$$

$$S(x)^D = A \cdot \eta[I(x)^U + [I(x)_M^D + I(x)^{\cancel{D}}]P(x)_{He}]$$

$$S(x)^D = \eta\,\underbrace{A[I(x)^U + I(x)^{\cancel{D}}\,P(x)_{He}]}_{S(x)^U} + \eta\,A\,\underbrace{I(x)_M^D}_{S(x)_M^D/B\eta P(x)_M}\,P(x)_{He}$$

$$S(x)^D = \eta\,[S(x)^U + \frac{A}{B} \cdot S(x)_M^D \frac{P(x)_{He}}{P(x)_M}$$

Using a A/B ratio of 1 and $\eta = 1 - C \cdot N^{2/3}$ the equations only depend on the position sensitive signal $S(x)$, which is determined by measuring the beam profiles, $P(x)_{He}$ and $P(x)_M$. While $P(x)_M$ is the term the equation is to be solved for $P(x)_{He}$ and the fact that $S(x)$ terms are accounting for all helium ionization channels still poeses problems. The first assumption to take to be able to solve for $P(x)_M$ is, that even though all ion channels form He_n contribute to $S(x)$, their intensities are very low compared to the ion channel of He_2. A decent linear approximation can therefore be calculated via extracting the size dependent ratio of He_2 to He_n from the mass spectra.

$$S(x)^D = \gamma \cdot S(x)_{He_2}^D$$

$$S(x)^U = \gamma \cdot S(x)_{He_2}^U$$

This and the assumption that the formation of He_2^+ is the dominant ionization cannel ($P(x)_{He} \sim 1$) gives the final fitting equation

$$S(x)_{He_2}^D = \left[1 - C \cdot N^{2/3}\right] \cdot S(x)_{He_2}^U + \frac{S(x)_M^D}{\gamma \cdot P(x)_M}$$

This equation is not considering fragmentation yet but can be modified to do so by factoring $S(x)_M^D$ according to the fragmentation pattern of the molecule.

The equations derived in this paragraph can therefore be used to approximate the charge transfer probability inside helium nanodroplets. By comparing the experimentally evaluated probability with the results calculated via different models could be used to settle on a certain charge transfer model.

$$\boldsymbol{P(x)_M \approx\ P(x, He_n^+ M \rightarrow M^+) \propto P_?(x, He_n^+ M \rightarrow M^+)}$$

Because of measurement inaccuracies and high errors corresponding to the simulation the evaluation of the gathered data for CsI and DMSO did not yield conclusive results yet. Improvements of the experimental setup, the evaluation software and simulation code are being conducted to improve the results.

Improvements to the experimental setup and evaluation

The first measurements of the charge transfer probability where carried out by Danial Merthe as part of his PhD thesis [3]. The experimental setup stayed basically the same since then, with the exemption of a new device to introduce liquid samples into the pickup chambers. A major update is on its way though. The quadrupole mass spectrometer will be upgraded to a new one, the primary improvements are a higher transition rate, better resolution and larger mass range. A digital interface is going to allow further automatization and easier accessibility. The ion detection will also be enhanced with the switch form analog current measurement to digital ion counting, which should yield way better signal to noise ratios especially for low beam intensities.

The new mass spectrometer is ready for testing, but further changes are needed to incorporate the new detection method into the evaluation and data acquisition software. The software was already subjected to significant changes in order to automate the repetitive tasks involved evaluating a measurement. The fitting algorithm was changed from Mathematica to python scripting language to easily interface with the simulation programs already used. The class based nonlinear fitting library used, brings some important improvements to the table. Because there is no restriction on the fitting function and a dynamic parameter space, the algorithm can handle multiple overlapping peaks and is able to introduce a custom peak shape by convoluting the gaussian model function with a spline interpolation. Although the custom peak shape is still work in progress, the advanced fitting library accompanied with a graphical user interface not only automates the fitting to a higher degree than before but also enhances the usability.

When fully integrated, the custom peak shape should enhance the data used as input for the simulation code quite a bit.

Future updates should tackle the data storage management. A python compatible database format should be implemented (The Django framework standard would be SQL) instead saving the spectra and deflection saved as csv-files with a tabular file indicating the settings for every file. Because the parsing of large data is leaves always a huge margin for error. Furthermore, should the evaluation software be directly interlinked with the simulation. To accomplish this, the simulation code has to be upgraded to at least python 3.6.

Measurements

The measurement campaign produced two sets of Cesium Iodide profiles and seven sets of DMSO profiles. An overview of the collected data is given in Appendix B. Both species are still under investigation.

Conclusion

The measurements of cesium iodide and DMSO have potential to reveal new information on the internal charge transfer mechanism associated with the ionization of impurities inside helium nanodroplets after electron bombardment. Due to the fact that the calculation of the parametrized charge transfer probability depends solely on the measured position dependent signal errors in those measurements are relayed quieted poorly. This combined with the similar parameterization of the different charge transfer models raises the need for a more accurate evaluation of the deflection profiles. This need is not fully satisfied yet. Even though the new fitting algorithm gives better reduced R values the custom peak shape feature has to be fully implemented to close the gap towards enough accuracy. The quality of the gathered data points seems appropriate but nevertheless will also be improved with the new quadrupole setup. These measures should render this experiment to an appropriate and unique tool to single out the most likely charge transfer model. The upgrade will furthermore also improve the experiments capabilities to measure electric dipole moments.

References

[1] W. Gerlach, Niels Bohr Archiev, Coppenhagen.

[2] H. Buchenau, E. L. Knut, J. Northby, J. P. Toennies and C. Winkler, "Mass spectra and time of flight distributions of helium cluster beams," *J. Chem. Phys.,* p. 6875, 1999.

[3] D. Methe, *Electric deflection of neutral doped helium nanodroplets,* Los Angeles, 2018.

[4] M. J. Buckingham and W. M. Fairbank, "The Nature of the Lambda-Transition in Liquid Helium," *Progress in Low Temperature Physics,* vol. 3, pp. 80-112, 1961.

[5] D. R. Tilley and J. Tilley, Superfluidity and Superconductivity, Bristol: IOP Publishing Ltd, 1990.

[6] S. Grebenev, J. P. Toennies and A. F. Vilesov, "Superfuidity within a small helium-4 cluster," *Science,* vol. 279, no. 5359, pp. 2083 - 2086, 1998.

[7] A. W. Hauser, A. Volk, P. Thaler and W. E. Ernst, "Atomic collisions in suprafluid helium-nanodroplets," *Phys. Chem. Chem. Phys.,* vol. 17, pp. 10805 - 10812, 2015.

[8] K. K. Lehmann and A. M. Dokter, "Evaporative Cooling of Helium Nanodroplets with Angular Momentum Conservation," *Phys. Rev. Lett.,* vol. 92, no. 17 - 30, p. 173401, 2004.

[9] S. Vongehr, T. Shao-Chun and M. Xiang-Kang, "Collision statistics of clusters: from Poisson model to Poisson mixture," *Chin. Phys. B,* vol. 19, no. 2, p. 023602, 2010.

[10] B. Shepperson, A. S. Chatterley, A. A. Sondergaard, L. L. M. Christiansen and H. Stapelfeldt, "Strongly aligned molecules inside helium droplets in the near-adiabatic regime," *The Journal of Chemical Physics,* vol. 147, p. 013946, 2017.

[11] D. V. Fursa and I. Bray, " Calculation of electron-helium scattering," *Phys. Rev. A,* vol. 52, no. 2, p. 1279, 1995.

[12] J. Harms, J. P. Toennies and F. Dalfovo, " Density of superfluid helium droplets," *Phys. Rev. B,* vol. 58, no. 6, p. 3341, 1998.

[13] W. K. Lewis, C. M. Lindsay, R. J. Bemish and R. M. Miller, " Probing chargetransfer processes in helium nanodroplets by optically selected mass spectrometry (OSMS)," *J. Am. Chem. Soc.,* vol. 127, no. 19, p. 7235, 2005.

[14] B. E. Callicoatt, D. D. Mar, V. A. Apkarian and K. C. Janda, " Charge transfer within He clusters," *J. Chem. Phys.,* vol. 105, no. 17, p. 7872, 1996.

[15] T. Ruchti, B. E. Callicoatt and K. C. Janda, " Charge transfer and fragmentation of liquid helium droplets doped with xenon," *Phys. Chem. Chem. Phys.,* vol. 2, no. 18, p. 4075, 2000.

[16] A. M. Ellis and S. Yang, "Model for the charge-transfer probability in helium nanodroplets following electron-impact ionization," *Phys. Rev. A,* vol. 76, no. 3, p. 032714, 2007.

[17] B. Bellina, D. J. Merthe and V. Kresin, "Proton transfer in histidine-tryptophan heterodimers embedded in helium droplets," *Journal of Chemical Physics,* vol. 142, p. 114306, 2015.

[18] W. Demtröder, Experimentalphysik 3: Atome, Moleküle und Festkörper, Berlin: Springer, 2010.

APPENDIX A

The following pages show the source code of the data evaluation tool (Python3.6):

```
from lmfit import minimize, Parameters, report_fit
from tkinter import filedialog
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2TkAgg
from matplotlib.figure import Figure
import tkinter as tk
from tkinter import simpledialog
import logging
import docx
import re
import os
import pandas as pd
import peakutils
import numpy as np
from scipy import interpolate
import weakref
import tkinter.scrolledtext as ScrolledText
from scipy.integrate import trapz


# Main Window
################################################################################
######################
class Main(tk.Tk):

    def __init__(self, *args, **kwargs):
        self.data = []
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, 'The Evaluator')
        container = tk.Frame(self)
        container.pack(side='top', fill='both', expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.active_fit = FittingWidget()

        button_frame0 = tk.Frame(self)
        button_frame0.pack(fill=tk.X)

        self.session0 = DataWidget()
        button0 = tk.Button(button_frame0, text="Browse", command=lambda:
self.session0.run_word())
        button_frame0.columnconfigure(1, weight=1)
        button0.grid(row=0, column=1, sticky=tk.W + tk.E)

        def func0(value):
            self.option0 = str(value)

        self.OPTIONS0 = ["Pressure", "Temperature", "Deflections", "Manual"]
        self.option0 = str("Pressure")
        self.variable0 = tk.StringVar(self)
        self.variable0.set(self.OPTIONS0[0])  # default value
        self.coordinates = []
        self.marker0 = None
        button1 = tk.OptionMenu(button_frame0, self.variable0, *self.OPTIONS0,
command=func0)
        button_frame0.columnconfigure(0, weight=1)
        button1.grid(row=0, column=0, sticky=tk.W + tk.E)

        def func1(value):
            self.option1 = str(value)

        self.OPTIONS1 = ["No Data Set Loaded", ]
        self.option1 = None
```

```
        self.variable1 = tk.StringVar(self)
        self.variable1.set(self.OPTIONS1[0])  # default value
        self.button2 = tk.OptionMenu(button_frame0, self.variable1, *self.OPTIONS1,
command=func1)
        button_frame0.columnconfigure(2, weight=1)
        self.button2.grid(row=0, column=2, sticky=tk.W + tk.E)

        self.Fig0 = Figure(figsize=(5, 5), dpi=100)
        self.ax0 = self.Fig0.add_subplot(1, 1, 1)
        self.canvas0 = FigureCanvasTkAgg(self.Fig0, self)
        toolbar = NavigationToolbar2TkAgg(self.canvas0, self)
        toolbar.update()
        FigureCanvasTkAgg.draw(self.canvas0)
        self.canvas0.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)
        self.canvas0._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        def onclick(event):
            try:
                self.marker0.remove()
            except:
                pass
            if event.dblclick:
                self.coordinates.append([event.xdata, event.ydata])
                self.marker0 = self.ax0.axvline(self.coordinates[-1][0],
linestyle='--')
                FigureCanvasTkAgg.draw(self.canvas0)
            else:
                pass

        self.Fig0.canvas.mpl_connect('button_press_event', onclick)

        def func2(value):
            inst = app.data[app.session0.temp.index(app.variable1.get())]
            inst.threshold = value
            try:
                inst.maxima = FittingWidget.find_peak_max(inst.axis, inst.signal,
self.scaler0.get(), 0.5)
            except:
                pass
            inst.show_data()

        scaler_frame0 = tk.Frame(self)
        scaler_frame0.pack(fill=tk.X)
        self.scaler0 = tk.Scale(self, from_=0, to=1, length=600, resolution=0.01,
orient='horizontal', label='Threshold', command=lambda x: func2(x))
        scaler_frame0.columnconfigure(0, weight=1)
        self.scaler0.pack()

        button_frame1 = tk.Frame(self)
        button_frame1.pack(fill=tk.X)

        button0 = tk.Button(button_frame1, text="Add", command=lambda:
app.data[app.session0.temp.index(app.variable1.get())].add_maxima(self.coordinates)
)
        button_frame1.columnconfigure(0, weight=1)
        button0.grid(row=0, column=0, sticky=tk.W + tk.E)

        button1 = tk.Button(button_frame1, text="Remove", command=lambda:
app.data[app.session0.temp.index(app.variable1.get())].remove_maxima(self.coordinat
es))
        button_frame1.columnconfigure(1, weight=1)
        button1.grid(row=0, column=1, sticky=tk.W + tk.E)

        button22 = tk.Button(button_frame1, text="Fit", command=lambda:
app.data[app.session0.temp.index(app.variable1.get())].run_fit())
        button_frame1.columnconfigure(2, weight=1)
        button22.grid(row=0, column=2, sticky=tk.W + tk.E)

        button3 = tk.Button(button_frame1, text="Set", command=lambda:
```

```
app.data[app.session0.temp.index(app.variable1.get())].choose_peak(self.coordinates
))
        button_frame1.columnconfigure(3, weight=1)
        button3.grid(row=0, column=3, sticky=tk.W + tk.E)

        def set_all():
            for name in self.OPTIONS1:
                try:

app.data[app.session0.temp.index(name)].choose_peak(self.coordinates)
                except:
                    pass

        button4 = tk.Button(button_frame1, text="Set All", command=lambda:
set_all())
        button_frame1.columnconfigure(4, weight=1)
        button4.grid(row=0, column=4, sticky=tk.W + tk.E)
        self.common = None
        def fit_all():
            app.common =
len(app.data[app.session0.temp.index(app.variable1.get())].maxima)
            for name in self.OPTIONS1[1:]:
                app.variable1.set(name)

app.data[app.session0.temp.index(app.variable1.get())].find_number_peaks(int(self.c
ommon))
                app.data[app.session0.temp.index(app.variable1.get())].run_fit()

        button5 = tk.Button(button_frame1, text="Fit All", command=lambda:
fit_all())
        button_frame1.columnconfigure(5, weight=1)
        button5.grid(row=0, column=5, sticky=tk.W + tk.E)

        self.pop0 = None

        def popup0():
            try:
                self.pop0.update_pop0()
            except:
                self.pop0 = PopupWidget0()

        button6 = tk.Button(button_frame1, text="EVAL VP", command=lambda:
popup0())
        button_frame1.columnconfigure(6, weight=1)
        button6.grid(row=0, column=6, sticky=tk.W + tk.E)

        self.pop1 = None

        def popup1():
            try:
                self.pop1.update_pop1()
            except:
                self.pop1 = PopupWidget1()

        button7 = tk.Button(button_frame1, text="EVAL FIT", command=lambda:
popup1())
        button_frame1.columnconfigure(7, weight=1)
        button7.grid(row=0, column=7, sticky=tk.W + tk.E)

        # Add text widget to display logging info
        self.st0 = ScrolledText.ScrolledText(self)
        self.st0.configure(font='TkFixedFont')
        self.st0.pack(fill=tk.X)

        # Create textLogger
        self.text_handler = LoggerWidget(self.st0)
        # Logging configuration
        logging.basicConfig(filename='test.log', level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')
```

```
        # Add the handler to logger
        self.logger = logging.getLogger()
        self.logger.addHandler(self.text_handler)


# tk.inter Widgets
################################################################################
##################
class LoggerWidget(logging.Handler):
    def __init__(self, textwidget):
        logging.Handler.__init__(self)
        self.setLevel(logging.DEBUG)
        self.widget = textwidget
        self.widget.config(state='normal')
        self.widget.tag_config("INFO", foreground="black")
        self.widget.tag_config("DEBUG", foreground="grey")
        self.widget.tag_config("WARNING", foreground="orange")
        self.widget.tag_config("ERROR", foreground="red")
        self.widget.tag_config("CRITICAL", foreground="red", underline=1)
        self.red = self.widget.tag_configure("red", foreground="red")

    def emit(self, record):
        self.widget.config(state='normal')
        # Append message (record) to the widget
        self.widget.insert(tk.END, self.format(record) + '\n', record.levelname)
        self.widget.see(tk.END)  # Scroll to the bottom
        self.widget.config(state='disabled')
        self.widget.update()  # Refresh the widget


class DataWidget(tk.Tk):
    def __init__(self):
        self.path = None
        self.files = None
        self.vector = []
        self.temp = []

    def run_word(self):
        self.vector = []
        self.path = filedialog.askopenfilename(filetypes=(('Word Document',
('docx', 'doc')), ('All', '*'),))
        logging.info('File directory: ' + str(os.path.split(self.path)[0]))
        self.run_files()

    def run_files(self):
        self.files = list(filedialog.askopenfilenames(filetypes=(('TSV', 'tsv'),
('All', '*'),)))
        self.pars_vector()
        self.update_menu()

    def update_menu(self):
        app.data = []

        def func(para):
            app.variable1.set(para)
            self.temp = []
            for entry in self.vector:
                self.temp.append(entry[0])
            app.data[self.temp.index(app.variable1.get())].show_data()
        app.variable1.set('Data Loaded')
        app.button2['menu'].delete(0, 'end')
        for item in self.vector:
            app.button2['menu'].add_command(label=item[0], command=lambda
x=item[0]: func(x))
            app.OPTIONS1.append(item[0])
            app.data.append(LogicWidget(item[0]))

    def pars_vector(self):
        doc = docx.Document(self.path)
        if app.option0 == "Pressure":
```

```
            print('Pressure')
            for item in self.files:
                j = 0
                for row in doc.tables[1].rows:
                    test0 = re.search((os.path.split(item)[1])[:2],
str(doc.tables[1].rows[j].cells[0].text))
                    if test0 is not None:
                        test1 = re.search('(Pcell = {0,3})([0-9]\.[0-9]{0,2}E-[0-
9])', str(doc.tables[1].rows[j].cells[5].text))
                        if test1 is not None:
                            self.vector.append([os.path.split(item)[1],
float(test1.group(2))])
                    j += 1
            logging.info(str(len(self.vector))+'/'+str(len(self.files))+' Files
where loaded successfully')

        if app.option0 == "Deflections":
            print('deflections')
            for item in self.files:
                j = 0
                table = 2
                for row in doc.tables[table].rows:
                    test0 = re.search((os.path.split(item)[1])[:2],
str(doc.tables[table].rows[j].cells[0].text))
                    if test0 is not None:
                        test1 = re.search('([0-9]{0,3})(.kV)',
str(doc.tables[table].rows[j].cells[5].text))
                        if test1 is not None:
                            self.vector.append([os.path.split(item)[1],
float(test1.group(1))])
                    j += 1
            logging.info(str(len(self.vector)) + '/' + str(len(self.files)) + '
Files where loaded successfully')

        if app.option0 == "Temperature":
            print('temperature')
            for item in self.files:
                j = 0
                table = 3
                for row in doc.tables[table].rows:
                    test0 = re.search((os.path.split(item)[1])[:2],
str(doc.tables[table].rows[j].cells[0].text))
                    if test0 is not None:
                        test1 = re.search('(Tcell = {0,3})([0-9]{0,3}) C',
str(doc.tables[table].rows[j].cells[5].text))
                        if test1 is not None:
                            self.vector.append([os.path.split(item)[1],
float(test1.group(2))])
                    j += 1
            logging.info(str(len(self.vector)) + '/' + str(len(self.files)) + '
Files where loaded successfully')

        if app.option0 == "Manual":
            print('manual')
            for item in self.files:
                answer = simpledialog.askstring(os.path.split(item)[1], 'Manual
Input')
                self.vector.append([os.path.split(item)[1], answer])


class LogicWidget:
    ID = 0
    vector = None
    pressure_point = []
    pressure_value = []
    mean_value = []
    mean_value = []
    area_value = []
    width_value = []
    name_vector = []
```

```
    export = pd.DataFrame()

    def __init__(self, para):
        self.ID = LogicWidget.ID
        LogicWidget.ID = LogicWidget.ID + 1
        self.name = para

        # print(self.name)
        self.file = None
        self.axis = None
        self.signal = None
        self.threshold = 0.05
        self.maxima = None
        self.points = []
        self.minima = None
        self.area = None
        self.base_spline = None
        self.y = None
        self.peaks = None


    def get_file(self):
        self.file = pd.read_csv(os.path.split(app.session0.path)[0] + "/" +
app.variable1.get(), sep='\t', header=None, usecols=[0, 1])

    def show_data(self):
        if self.file is None:
            self.get_file()
            self.axis = np.array(self.file.iloc[:, 0].values)
            self.signal = np.array(self.file.iloc[:, 1].values)
            self.maxima = FittingWidget.find_peak_max(self.axis, self.signal,
app.scaler0.get(), 0.5)
        app.scaler0.set(self.threshold)
        app.ax0.clear()
        app.ax0.scatter(self.axis[self.maxima], self.signal[self.maxima])
        app.ax0.set_xlim(min(self.axis), max(self.axis))
        app.ax0.set_ylim(min(self.signal), max(self.signal)*1.2)
        app.ax0.plot(self.axis, self.signal)
        try:
            app.ax0.scatter(self.axis[self.points], self.signal[self.points])
        except:
            pass
        try:
            for item in self.peaks:
                a = item[0]
                b = item[1]
                c = item[2]
                xspace = np.linspace(min(self.axis), max(self.axis), 10000)
                app.ax0.plot(xspace, a * np.exp(-np.power(xspace - b, 2.) / (2 *
np.power(c, 2.))), linestyle=':')
        except:
            pass
        FigureCanvasTkAgg.draw(app.canvas0)

    def add_maxima(self, coordinates):
        try:
            self.maxima = np.array(np.append(self.maxima, np.argmin(abs(self.axis -
float(coordinates[-1][0]))))).astype(int)
            self.show_data()
        except:
            logging.warning('COULD NOT ADD POINT')

    def remove_maxima(self, coordinates):
        try:
            temp = abs(self.axis[self.maxima] - float(coordinates[-1][0]))
            self.maxima = np.delete(self.maxima, np.argmin(temp))
            self.show_data()
        except:
            logging.warning('COULD NOT REMOVE POINT')
```

```
    def find_number_peaks(self, length):
        app.data[app.session0.temp.index(app.variable1.get())].show_data()
        app.scaler0.set(self.threshold)
        self.maxima = FittingWidget.find_peak_max(self.axis, self.signal,
app.scaler0.get(), 0.5)
        if len(self.maxima) > length:
            self.threshold = str("%.2f" % (float(self.threshold) + 0.005))
            # print(self.threshold)
            LogicWidget.find_number_peaks(self, length)
        if len(self.maxima) < length:
            self.threshold = str("%.2f" % (float(self.threshold) - 0.005))
            # print(self.threshold)
            LogicWidget.find_number_peaks(self, length)
        if self.threshold == 0:
            logging.warning('Min Threshhold reached')

    def choose_peak(self, coordinates):
        try:
            app.ax0.remove(self.set)
        except:
            pass
        self.mean = []
        for item in self.peaks:
            self.mean.append(item[1])
        self.set_value = np.argmin(abs(self.mean - coordinates[-1][0]))
        self.show_data()
        a = (self.peaks[self.set_value][0])
        b = (self.peaks[self.set_value][1])
        c = (self.peaks[self.set_value][2])
        xspace = np.linspace(min(self.axis), max(self.axis), 10000)
        peak = a * np.exp(-np.power(xspace - b, 2.) / (2 * np.power(c, 2.)))
        self.area = trapz(peak, xspace)
        self.pos = b
        self.width = c

        LogicWidget.export.loc[:, (str(self.name)+' x values')] = xspace
        LogicWidget.export.loc[:, (str(self.name)+' y Values')] = peak
        LogicWidget.export.to_csv(os.path.split(app.session0.path)[0]+'/out.tsv',
sep='\t', index=False)
        print(LogicWidget.export)
        logging.info('File Saved as
'+str(os.path.split(app.session0.path)[0]+'/out.tsv'))

    def run_fit(self):
        self.peaks = FittingWidget.fitting(app.active_fit, [self.axis, self.signal,
self.maxima])


class FittingWidget:

    @staticmethod
    def find_peak(y, t, d):
        data_max = peakutils.indexes(y, thres=t, min_dist=d)
        return [data_max]

    @staticmethod
    def find_peak_max(x, y, d=0.5, t=0.0005, z=None):
        peak_max = []
        if z is not None:
            peak_max.append(0)
        temp = (FittingWidget.find_peak(y, d, t)[0])
        for i in range(len(temp)):
            peak_max.append(temp[i])
        if z is not None:
            peak_max.append(len(x) - 1)
        return peak_max

    @staticmethod
    def get_base(y):
        base = peakutils.baseline(y, 1)
```

```
        return base

    @staticmethod
    def find_guess(x, y, pos_max):
        guess = []
        c = 0
        dx = 25
        for item in pos_max:
            j = 0
            while True:
                if x[item+j] == x[-1]:
                    print('Peak', c, 'loop break => x right out of bounds')
                    break
                if x[item-j] == x[0]:
                    print('Peak', c, 'loop break => x left out of bounds')
                    break
                try:
                    if y[item+j] <= min(y[item+j-dx:item+j+dx]):
                        print('Peak', c, 'loop break => local minimum at',
x[item+j])
                        break
                except:
                    pass

                if y[item]*.60 <= y[item + j]:
                    j += 1
                else:
                    print('Peak', c, '60% border width is', x[item+j]-x[item-j])
                    print('Border Points are', x[item-j], x[item+j])
                    break

            guess.append((trapz(y[item-j:item+j])/100, x[item], (x[item+j] -
x[item-j])/2, x[item-j], x[item+j]))
            c += 1

        return guess

    @staticmethod
    def gauss(params, x, data, pos_max):
        func = []
        j = 1
        for item in pos_max:
            alpha = params['n'+str(j)].value
            j += 1
            bravo = params['n'+str(j)].value
            j += 1
            charlie = params['n'+str(j)].value
            j += 1

            func.append(alpha * np.exp(-np.power(x - bravo, 2.) / (2 *
np.power(charlie, 2.))))

        return sum(func) - data

    def fitting(self, para):
        xdata = para[0]
        ydata = para[1]
        pos_max = para[2]
        guess = FittingWidget.find_guess(xdata, ydata, pos_max)

        params = Parameters()
        j = 1
        for item in guess:
            params.add('n'+str(j), value=item[0], min=0)
            j += 1
            params.add('n'+str(j), value=item[1], min=item[3], max=item[4])
            j += 1
            params.add('n'+str(j), value=item[2], min=0.05, max=item[2]*1.25)
            j += 1
        result = minimize(self.gauss, params, args=(xdata, ydata, pos_max))
```

```
        self.xplot = xdata
        self.yplot = self.gauss(result.params, xdata, ydata, pos_max) + ydata
        report_fit(result)

        logging.info(str(''))

logging.info(str('################################Leastsquare###################
###############'))
        logging.info(str(''))
        logging.info(str('Function Evaluations: ') + str(result.nfev))
        logging.info(str('Number of Variables: ') + str(result.nvarys))
        logging.info(str('Chi squared: ') + str(result.chisqr))
        logging.info(str('Reduced Chi squ: ') + str(result.redchi))
        logging.info(str('Akaike info Crit: ') + str(result.aic))
        logging.info(str('Bayesian info Crit: ') + str(result.bic))
        logging.info(str(''))

logging.info(str('################################################################
###############'))
        logging.info(str(''))

        self.peaks = []
        xspace = np.linspace(min(xdata), max(xdata), 10000)
        j = 1
        for item in pos_max:
            a = result.params['n' + str(j)].value
            j += 1
            b = result.params['n' + str(j)].value
            j += 1
            c = result.params['n' + str(j)].value
            j += 1
            self.peaks.append([a, b, c])
            app.ax0.plot(xspace, a * np.exp(-np.power(xspace - b, 2.) / (2 *
np.power(c, 2.))), linestyle=':')
        FigureCanvasTkAgg.draw(app.canvas0)
        return self.peaks

    def __init__(self):
        self.xdata = None
        self.ydata = None
        self.xplot = None
        self.yplot = None
        self.pos_max = None
        self.data_max = None


class PopupWidget0(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, 'Vapor Pressure Curve')
        container = tk.Frame(self)
        container.pack(side='top', fill='both', expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        j = 0
        LogicWidget.pressure_value = []
        LogicWidget.pressure_point = []
        for item in app.data:
            LogicWidget.pressure_point.append([app.session0.vector[j][1]])
            LogicWidget.pressure_value.append([app.data[j].area])
            j += 1

        self.Fig0 = Figure(figsize=(5, 5), dpi=100)
        self.ax0 = self.Fig0.add_subplot(1, 1, 1)
        self.ax0.set_xlim((*min(LogicWidget.pressure_point),
*max(LogicWidget.pressure_point)))
        print(min(*LogicWidget.pressure_point))
        print(max(*LogicWidget.pressure_point))
        self.canvas0 = FigureCanvasTkAgg(self.Fig0, self)
```

```
        toolbar = NavigationToolbar2TkAgg(self.canvas0, self)
        toolbar.update()
        FigureCanvasTkAgg.draw(self.canvas0)
        self.canvas0.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)
        self.canvas0._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
        self.ax0.scatter(LogicWidget.pressure_point, LogicWidget.pressure_value)
        FigureCanvasTkAgg.draw(self.canvas0)

    def update_pop0(self):
        j = 0
        LogicWidget.pressure_value = []
        LogicWidget.pressure_point = []
        for item in app.data:
            LogicWidget.pressure_point.append([app.session0.vector[j][1]])
            LogicWidget.pressure_value.append([app.data[j].area])
            j += 1
        self.ax0.clear()
        self.ax0.set_xlim((*min(LogicWidget.pressure_point),
*max(LogicWidget.pressure_point)))
        self.ax0.scatter(LogicWidget.pressure_point, LogicWidget.pressure_value)
        FigureCanvasTkAgg.draw(self.canvas0)


class PopupWidget1(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, 'Fit Parameter')
        container = tk.Frame(self)
        container.pack(side='top', fill='both', expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        j = 0
        LogicWidget.mean_value = []
        LogicWidget.area_value = []
        LogicWidget.width_value = []
        LogicWidget.name_vector = []
        for item in app.data:
            LogicWidget.mean_value.append([app.data[j].pos])

            LogicWidget.area_value.append([app.data[j].area])

            LogicWidget.width_value.append([app.data[j].width])

            LogicWidget.name_vector.append([app.data[j].name])
            j += 1
        self.Fig0 = Figure(figsize=(5, 5), dpi=100)
        self.ax0 = self.Fig0.add_subplot(3, 1, 1)
        self.ax1 = self.Fig0.add_subplot(3, 1, 2)
        self.ax2 = self.Fig0.add_subplot(3, 1, 3)
        self.ax0.set_ylabel('Mean')
        self.ax1.set_ylabel('Area')
        self.ax2.set_ylabel('Width')
        self.ax0.set_yticklabels([])
        self.ax0.set_yticks([])
        self.ax1.set_yticklabels([])
        self.ax1.set_yticks([])
        self.ax2.set_yticklabels([])
        self.ax2.set_yticks([])
        self.canvas0 = FigureCanvasTkAgg(self.Fig0, self)
        self.coordinates1 = []
        toolbar = NavigationToolbar2TkAgg(self.canvas0, self)
        toolbar.update()
        FigureCanvasTkAgg.draw(self.canvas0)
        self.canvas0.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH,
expand=True)
        self.canvas0._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        self.ax0.set_xlim((*min(LogicWidget.mean_value),
```

```
*max(LogicWidget.mean_value)))
        self.ax0.scatter(LogicWidget.mean_value, range(0,
len(LogicWidget.mean_value)))
        j = 0
        for point in LogicWidget.mean_value:
            self.ax0.annotate(*LogicWidget.name_vector[j], xy=(*point, j+0.2))
            j += 1

        self.ax1.set_xlim((*min(LogicWidget.area_value),
*max(LogicWidget.area_value)))
        self.ax1.scatter(LogicWidget.area_value, range(0,
len(LogicWidget.area_value)))
        j = 0
        for point in LogicWidget.area_value:
            self.ax1.annotate(*LogicWidget.name_vector[j], xy=(*point, j+0.2))
            j += 1

        self.ax2.set_xlim((*min(LogicWidget.width_value),
*max(LogicWidget.width_value)))
        self.ax2.scatter(LogicWidget.width_value, range(0,
len(LogicWidget.width_value)))
        j = 0
        for point in LogicWidget.width_value:
            self.ax2.annotate(*LogicWidget.name_vector[j], xy=(*point, j+0.2))
            j += 1
        FigureCanvasTkAgg.draw(self.canvas0)

        def onclick(event):
            try:
                self.marker1.remove()
            except:
                pass
            if event.dblclick:
                self.coordinates1.append([event.xdata, event.ydata])
                self.marker1 = self.ax0.axvline(self.coordinates1[-1][0],
linestyle='--')
                FigureCanvasTkAgg.draw(self.canvas0)
                self.ax1.scatter(LogicWidget.mean_value, range(0,
len(LogicWidget.mean_value)))
            else:
                pass

        self.Fig0.canvas.mpl_connect('button_press_event', onclick)

    def update_pop1(self):
        j = 0
        LogicWidget.mean_value = []
        LogicWidget.name_vector = []
        for item in app.data:
            LogicWidget.mean_value.append([app.data[j].parameter])
            LogicWidget.name_vector.append([app.data[j].name])
            j += 1
        self.ax0.clear()
        self.ax0.set_xlim((*min(LogicWidget.mean_value),
*max(LogicWidget.mean_value)))
        self.ax0.scatter(LogicWidget.mean_value, LogicWidget.name_vector)
        FigureCanvasTkAgg.draw(self.canvas0)


app = Main()
app.mainloop()
```