

Fano Resonances in Time-Dependent Wells

Author: Anton Gregefalk and Supervisor: Tanay Nag

(Dated: April 12, 2023)

Floquet's theorem, a temporal analogue of Bloch's theorem, is used for studying a time-dependent potential. With applications in cold atoms on optical lattices, quantum dots and more, there is a growing interest in Floquet engineering exotic materials and phases. By solving the time-dependent Schrödinger equation scattering amplitudes are derived from which the transmission spectrum are generated. The driving field induces Floquet sidebands into which the particle can inelastically scatter. Fano resonances are observed when the incoming particle and a bound state of the static potential differ by some energy quanta. This process is mediated by the driving field. The scattering matrix and transmission spectra are reproduced from previous work on electron gas, graphene, and a semi-metal admitting a point of quadratic band touching (QBT). The QBT system is extended with linear tilting along the potential, which proves to be another good quantum number for tunable control.

I. INTRODUCTION

Understanding electronic properties are of fundamental interest and importance in physics and for society. For instance, semiconductors such as transistors are used in most electronic devices today, where the intricate design and manipulation of the band gap is paramount. The processes of interest are typically triggered by non-equilibrium light-matter interactions. Moreover, applying external electromagnetic fields to the system can alter these properties. This, sometimes drastic, response enables control of certain material properties including transport. By continuously irradiating a material the effective behaviour can be modelled by a time-periodic Hamiltonian. In the non-adiabatic regime the system can be studied through the lens of Floquet(-Bloch) Theory, a complete temporal analogue of the Bloch-theory for spatially periodic systems (crystals). The result is an analytical framework for studying scattering properties and how they change due to tunable parameters of the driving. Notably, Fano resonances occur by mediation of the driving field when an evanescent mode of the charge carrier overlaps with a bound state mode of the static potential. The techniques are being harnessed for so called Floquet Engineering, with most of the application so far taking place in systems of ultra-cold atoms in optical traps. Several exotic quantum materials are also being studied via this framework, such as band topology deformations, improved spintronics, as well as extending theoretical tools [1].

In this report the focus is on single charge carriers interacting with a rectangular potential well of depth V_0 and width L and a driving field with frequency ω and amplitude V_1 contained in the same region. The potential is assumed to be strictly along the x -direction, with transversal invariance in other directions. Explicitly this is

$$V(x, t) = \begin{cases} V_0 + V_1 \cos(\omega t) & , \text{ if } x \in [-L/2, L/2] \\ 0 & , \text{ elsewhere} \end{cases}$$

It is possible to study any number of oscillators, but we restrict ourselves to the simplest case of a single field of

sinusoidal character still providing a rich subject. The Floquet S-matrix is derived from boundary conditions of the solutions to the Schrödinger equation. When normalized it is the scattering matrix containing information about transmission and reflection amplitudes. The transmission spectra are generated for the cases of an incident beam from the left propagating to the right, through the potential. Experimentally this can be studied via the conductance for some cases, however there might be systems where the behaviours are too sensitive to be captured by such means, therefore the shot-noise is also generated. Any parameters used are presented in A found in appendix A, and the python codes constructed are found in appendix B.

To get familiar with the methods and formalism, previous results are reproduced. Section II covers the work of Li and Reichl [2], where the basic case of a 1-dimensional system is studied. Here the groundwork for working in the Floquet formalism is established, so that similar details can be skipped in later sections. The suggested mechanism behind the Fano resonances are also discussed. Section III explains pumped shot-noise, as used in Refs.[3, 4], and how it is used for detecting resonances. Sections IV and VI covers the work of Zhu et. al. [3] which extends the previous section to a 2-dimensional electron gas (2DEG) as well as the 2-level graphene. Section IV is purely mathematical, where a general result for the wavefunctions of a two-level system is derived by diagonalizing a general 2×2 Hamiltonian. Section VII is the last review section, covering the work of Bera and Mandal [4] regarding a 2D, 2-level, Weyl-semimetal with a quadratic band touching point (QBT). Such a structure can be realised in checker-board, kagome, and Lieb lattices at different fillings. Our contribution is contained in Section VIII. Here we extend the QBT Hamiltonian to include $\tau_y p_y$, tilting the band structure along y . We compare the effect of tilt to the previous section. At the end we finish with some concluding remarks and suggestions for further research.

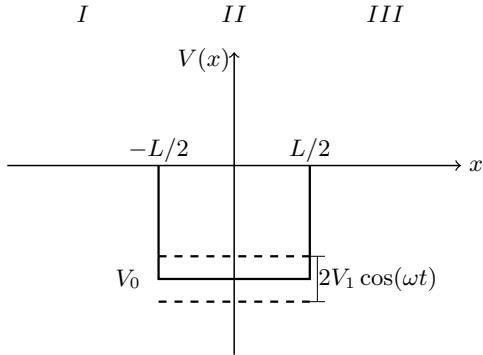


FIG. 1: Along x the space is divided into three regions. Regions I and III are to the left, $-\infty < x < -L/2$ and right, $L/2 < x < \infty$ of the potential, respectively. Here the potential is zero and the propagator is a free particle. In region II, $-L/2 \leq x \leq L/2$, there is the electrostatic potential V_0 and the dynamic contribution $2V_1 \cos(\omega t)$. The system is assumed homogeneous along y , providing transversal invariance.

II. 1-DIMENSIONAL CASE

For one incident non-relativistic electron with effective mass μ the Hamiltonian is

$$H = -\frac{\hbar^2}{2\mu} \frac{\partial^2}{\partial x^2} + V(x, t),$$

for the time-dependent Schrödinger equation (TDSE)

$$i\hbar \partial_t \Psi = H\Psi.$$

The space can be organised into three regions, pictured in Fig.[1], with regions I and III being the free space outside the potential and region II being the extension of the potential. With the presence of a strong enough oscillating potential Floquet's Theorem [5] ensures a solution of the form

$$\psi_F(x, t) = e^{-iE_F t/\hbar} \phi(x, t) \quad (1)$$

where E_F is the Floquet quasi-energy taken to be the Fermi energy proceeding forward, and $\phi(x, t + T) = \phi(x, t)$ is periodical with period $T = 2\pi/\omega$, ω is the driving frequency. This is in complete analogy with Bloch's theorem for a spatial periodicity, i.e. changing the time-energy interplay to position-momentum would recreate exactly that of Bloch. Hence the theory is sometimes called Floquet-Bloch theory. Plugging ψ_F into the TDSE we get

$$E_F \phi(x, t) + i\hbar \frac{\partial}{\partial t} \phi(x, t) = -\frac{\hbar^2}{2\mu} \frac{\partial^2}{\partial x^2} \phi(x, t) + V(x, t) \phi(x, t)$$

Starting with region II we notice the variables are independent, motivating a separation of variables approach, $\phi_{II}(x, t) = g(x)f(t)$. The TDSE then becomes

$$i\hbar \frac{1}{f(t)} \frac{\partial}{\partial t} f(t) - V_1 \cos(\omega t) + E_F = -\frac{\hbar^2}{2\mu} \frac{1}{g(x)} \frac{\partial^2}{\partial x^2} g(x) + V_0.$$

The two sides are dependent on different variables, meaning the only way for the DE's to have a solution is if they are equal to some constant E , meaning

$$\left\{ \begin{array}{l} i\hbar \frac{1}{f(t)} \frac{\partial}{\partial t} f(t) - V_1 \cos(\omega t) + E_F = E, \\ -\frac{\hbar^2}{2\mu} \frac{1}{g(x)} \frac{\partial^2}{\partial x^2} g(x) + V_0 = E. \end{array} \right. \quad (2)$$

$$\left\{ \begin{array}{l} i\hbar \frac{1}{f(t)} \frac{\partial}{\partial t} f(t) - V_1 \cos(\omega t) + E_F = E, \\ -\frac{\hbar^2}{2\mu} \frac{1}{g(x)} \frac{\partial^2}{\partial x^2} g(x) + V_0 = E. \end{array} \right. \quad (3)$$

The temporal equation (2) can be written as

$$\frac{\partial}{\partial t} f(t) = -\frac{i}{\hbar} (E - E_F) f(t) - \frac{iV_1}{\hbar} \cos(\omega t) f(t),$$

with solution

$$\begin{aligned} f(t) &= \exp \left(\int_0^t dt' \left[\frac{-i}{\hbar} (E - E_F) - \frac{iV_1}{\hbar} \cos(\omega t') \right] \right) \\ &= e^{-i(E-E_F)t/\hbar} e^{-iV_1 \sin(\omega t)/\hbar\omega} \\ &= e^{-i(E-E_F)t/\hbar} \sum_{n=-\infty}^{\infty} J_n \left(\frac{V_1}{\hbar\omega} \right) e^{-in\omega t}, \end{aligned} \quad (4)$$

where we have used the Jacobi-Anger identity and $J_n(x)$ are the normal type Bessel functions. The periodic constraint on $\phi(x, t)$ transfers to $f(t)$ meaning $f(t) = f(t + T) = f(t + 2\pi/\omega)$. This obviously holds in the frequency exponential. The energy exponential, however, gives

$$\begin{aligned} e^{-i(E-E_F)t/\hbar} &= e^{-i(E-E_F)(t+2\pi/\omega)/\hbar} \\ &= e^{-i(E-E_F)t/\hbar} e^{-i(E-E_F)2\pi/\hbar\omega} \end{aligned}$$

implying the second factor must vanish. This occurs when

$$-i(E-E_F)2\pi/\hbar\omega = i2\pi m \implies E = E_F + m\hbar\omega, \quad m \in \mathbb{Z},$$

giving us a condition on the mutual energy parameter E . Plugging this into the exponential gives

$$e^{-i(E-E_F)t/\hbar} = e^{-i(E_F + m\hbar\omega - E_F)t/\hbar} = e^{-im\omega t},$$

meaning (4) becomes

$$f(t) = \sum_{n=-\infty}^{\infty} J_n \left(\frac{V_1}{\hbar\omega} \right) e^{-i(n+m)\omega t},$$

which we write alternatively, by letting $n \rightarrow n - m$, as

$$f(t) = \sum_{n=-\infty}^{\infty} J_{n-m} \left(\frac{V_1}{\hbar\omega} \right) e^{-in\omega t}. \quad (5)$$

This modulation, entirely dependent on the driving strength and frequency, is what quantizes the energy for fixed E_F . By this token the energy landscape can be viewed as consisting of (Floquet-)sidebands or channels into which the propagator can scatter with probabilities weighted by the Bessel functions, depicted in Fig.[2]

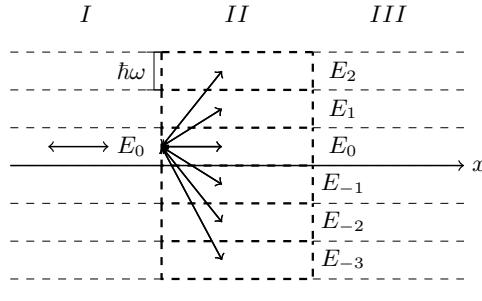


FIG. 2: The V_1 contribution enables this different view of the energy landscape. A particle propagating with energy E_n can jump energy level inside the potential to E_m with probability weighted by the Bessel function $J_{n-m}(V_1/\hbar\omega)$. The wavefunction carries a superposition of all possible level jumps.

Moving on to the spatial equation (3) we have

$$-\frac{\hbar^2}{2\mu} \frac{1}{g(x)} \frac{\partial^2}{\partial x^2} g(x) + V_0 = E$$

which we write as

$$\frac{\partial^2}{\partial x^2} g(x) = -\frac{2\mu}{\hbar^2} (E - V_0) g(x)$$

with general solution

$$g(x) = ae^{iqx} + be^{-iqx} \quad (6)$$

where we have defined

$$q^2 := \frac{2\mu}{\hbar^2} (E - V_0).$$

Since E is discrete in terms of m every such integer corresponds to a solution. Moreover any linear combination of them are solutions. We can thus extend (6) to

$$g(x) = \sum_{m=-\infty}^{\infty} (a_m e^{iq_m x} + b_m e^{-iq_m x}) \quad (7)$$

where we now note the energy as $E_m := E_F + m\hbar\omega$ and apply indices accordingly on the momenta as well as the probability amplitudes. The complete wavefunction for region II is thus

$$\begin{aligned} \psi_{II}(x, t) &= e^{-iE_F t/\hbar} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} (a_m e^{iq_m x} + b_m e^{-iq_m x}) J_{n-m}\left(\frac{V_1}{\hbar\omega}\right) e^{-in\hbar\omega t} \\ &= \sum_{n=-\infty}^{\infty} e^{-iE_n t/\hbar} \sum_{m=-\infty}^{\infty} (a_m e^{iq_m x} + b_m e^{-iq_m x}) J_{n-m}\left(\frac{V_1}{\hbar\omega}\right). \end{aligned} \quad (8)$$

Moving on to the outer regions. In a similar fashion, but for $V(x, t) = 0$ we simply get instead

$$k_n^2 := \frac{2\mu}{\hbar^2} E_n$$

and the infinite sum of the temporal part vanishes. We get for regions I and III

$$\begin{aligned} \psi_I(x, t) &= e^{-iE_F t/\hbar} \sum_{n=-\infty}^{\infty} (A_n^i e^{ik_n x} + A_n^o e^{-ik_n x}) e^{-in\hbar\omega t/\hbar} \\ &= \sum_{n=-\infty}^{\infty} e^{-iE_n t/\hbar} (A_n^i e^{ik_n x} + A_n^o e^{-ik_n x}), \end{aligned}$$

and

$$\psi_{III}(x, t) = \sum_{n=-\infty}^{\infty} e^{-iE_n t/\hbar} (B_n^o e^{ik_n x} + B_n^i e^{-ik_n x}),$$

respectively. The superscripts i and o of amplitudes A and B correspond to incoming and outgoing waves from

left and right respectively. Boundary conditions for a smooth continuous function give the following

$$\psi_I(-L/2, t) = \psi_{II}(-L/2, t) \quad (9)$$

$$\frac{\partial}{\partial x} \psi_I \Big|_{-L/2} = \frac{\partial}{\partial x} \psi_{II} \Big|_{-L/2} \quad (10)$$

$$\psi_{II}(L/2, t) = \psi_{III}(L/2, t) \quad (11)$$

$$\frac{\partial}{\partial x} \psi_{II} \Big|_{L/2} = \frac{\partial}{\partial x} \psi_{III} \Big|_{L/2}. \quad (12)$$

Noting that for the exponentials to match, they equal

only with the same index. Explicitly this gives, for (9)

$$\begin{aligned} & \sum_{n=-\infty}^{\infty} \left(A_n^i e^{-ik_n L/2} + A_n^o e^{ik_n L/2} \right) e^{-iE_n t/\hbar} \\ &= \sum_{n=-\infty}^{\infty} e^{-iE_n t/\hbar} \sum_{m=-\infty}^{\infty} (a_m e^{iq_m x} + b_m e^{-iq_m x}) J_{n-m} \left(\frac{V_1}{\hbar\omega} \right) \\ &\implies A_n^i e^{-ik_n L/2} + A_n^o e^{ik_n L/2} \\ &= \sum_{m=-\infty}^{\infty} \left(a_m e^{-iq_m L/2} + b_m e^{iq_m L/2} \right) J_{n-m} \left(\frac{V_1}{\hbar\omega} \right), \end{aligned} \quad (13)$$

and for (10)

$$\begin{aligned} ik_n A_n^i e^{-ik_n L/2} - ik_n A_n^o e^{ik_n L/2} &= \\ & \sum_{m=-\infty}^{\infty} (iq_m a_m e^{-iq_m L/2} - iq_m b_m e^{iq_m L/2}) J_{n-m} \left(\frac{V_1}{\hbar\omega} \right). \end{aligned} \quad (14)$$

Similarly, (11) and (12) becomes

$$\begin{aligned} B_n^o e^{ik_n L/2} + B_n^i e^{-ik_n L/2} &= \\ & \sum_{m=-\infty}^{\infty} (a_m e^{iq_m L/2} + b_m e^{-iq_m L/2}) J_{n-m} \left(\frac{V_1}{\hbar\omega} \right), \end{aligned} \quad (15)$$

and

$$\begin{aligned} ik_n B_n^o e^{ik_n L/2} - ik_n B_n^i e^{-ik_n L/2} &= \\ & \sum_{m=-\infty}^{\infty} (iq_m a_m e^{iq_m L/2} - iq_m b_m e^{-iq_m L/2}) J_{n-m} \left(\frac{V_1}{\hbar\omega} \right), \end{aligned} \quad (16)$$

respectively. Now we combine expressions as $ik_n(13) + (14)$ and $ik_n(15) - (16)$ to get

$$\begin{aligned} 2k_n A_n^i e^{-ik_n L/2} &= \sum_{m=-\infty}^{\infty} [(k_n + q_m) a_m e^{-iq_m L/2} \\ &+ (k_n - q_m) b_m e^{iq_m L/2}] J_{n-m} \left(\frac{V_1}{\hbar\omega} \right), \end{aligned} \quad (17)$$

and

$$\begin{aligned} 2k_n B_n^i e^{-ik_n L/2} &= \sum_{m=-\infty}^{\infty} [k_n - q_m] a_m e^{iq_m L/2} \\ &+ (k_n + q_m) b_m e^{-iq_m L/2} J_{n-m} \left(\frac{V_1}{\hbar\omega} \right), \end{aligned} \quad (18)$$

respectively. Finally we combine (17) \pm (18) to get

$$\begin{aligned} 2k_n (A_n^i \pm B_n^i) e^{-ik_n L/2} &= \\ & \sum_{m=-\infty}^{\infty} [(k_n + q_m) e^{-iq_m L/2} \\ & \pm (k_n - q_m) e^{iq_m L/2}] C_m^{\pm} J_{n-m} \left(\frac{V_1}{\hbar\omega} \right) \end{aligned} \quad (19)$$

where we have defined

$$C_m^{\pm} := a_m \pm b_m.$$

Next we define some matrix elements, and subsequently re-express our results in terms of matrices for a more general view.

$$\begin{aligned} (M_s^{\pm})_{nm} &= [(k_n + q_m) e^{-iq_m L/2} \\ & \pm (k_n - q_m) e^{iq_m L/2}] J_{n-m} \left(\frac{V_1}{\hbar\omega} \right) \\ (M_r)_{nm} &= 2k_n e^{-ik_n L/2} \delta_{nm} \\ (M_c^{\pm})_{nm} &= e^{-i(k_n \pm q_m)L/2} J_{n-m} \left(\frac{V_1}{\hbar\omega} \right) \\ (M_i)_{nm} &= e^{-ik_n L/2} \delta_{nm} \end{aligned}$$

We can now write (19) as

$$M_s^{\pm} \cdot \mathbf{C}^{\pm} = M_r \cdot (\mathbf{A}^i \pm \mathbf{B}^i)$$

Thus

$$\mathbf{C}^{\pm} = (M_s^{\pm})^{-1} \cdot M_r \cdot (\mathbf{A}^i \pm \mathbf{B}^i).$$

We can then express the well amplitudes as

$$\begin{aligned} \mathbf{a} &= \frac{1}{2} (\mathbf{C}^+ + \mathbf{C}^-) \\ &= \frac{1}{2} [(M_s^+)^{-1} + (M_s^-)^{-1}] \cdot M_r \cdot \mathbf{A}^i \\ &+ \frac{1}{2} [(M_s^+)^{-1} - (M_s^-)^{-1}] \cdot M_r \cdot \mathbf{B}^i \end{aligned}$$

and

$$\begin{aligned} \mathbf{b} &= \frac{1}{2} (\mathbf{C}^+ - \mathbf{C}^-) \\ &= \frac{1}{2} [(M_s^+)^{-1} - (M_s^-)^{-1}] \cdot M_r \cdot \mathbf{A}^i \\ &+ \frac{1}{2} [(M_s^+)^{-1} + (M_s^-)^{-1}] \cdot M_r \cdot \mathbf{B}^i \end{aligned}$$

Moving on to the outgoing amplitudes we have

$$\begin{aligned} \mathbf{A}^o &= M_c^+ \cdot \mathbf{a} + M_c^- \cdot \mathbf{b} - M_i \cdot \mathbf{A}^i \\ \mathbf{B}^o &= M_c^- \cdot \mathbf{a} + M_c^+ \cdot \mathbf{b} - M_i \cdot \mathbf{B}^i \end{aligned}$$

By plugging in the results for \mathbf{a} and \mathbf{b} we get

$$\begin{aligned}
\mathbf{A}^o &= \left\{ \frac{1}{2} (M_c^+ \cdot [(M_s^+)^{-1} + (M_s^-)^{-1}] + M_c^- \cdot [(M_s^+)^{-1} - (M_s^-)^{-1}]) \cdot M_r - M_i \right\} \cdot \mathbf{A}^i \\
&\quad + \frac{1}{2} (M_c^+ \cdot [(M_s^+)^{-1} - (M_s^-)^{-1}] + M_c^- \cdot [(M_s^+)^{-1} + (M_s^-)^{-1}]) \cdot M_r \cdot \mathbf{B}^i \\
\mathbf{B}^o &= \frac{1}{2} (M_c^+ \cdot [(M_s^+)^{-1} + (M_s^-)^{-1}] + M_c^- \cdot [(M_s^+)^{-1} - (M_s^-)^{-1}]) \cdot M_r \cdot \mathbf{A}^i \\
&\quad + \left\{ \frac{1}{2} (M_c^+ \cdot [(M_s^+)^{-1} - (M_s^-)^{-1}] + M_c^- \cdot [(M_s^+)^{-1} + (M_s^-)^{-1}]) \cdot M_r - M_i \right\} \cdot \mathbf{B}^i
\end{aligned}$$

To ease the notation further we define matrices such that the equations becomes

$$\begin{cases} \mathbf{A}^o = M_{AA} \cdot \mathbf{A}^i + M_{AB} \cdot \mathbf{B}^i, \\ \mathbf{B}^o = M_{BA} \cdot \mathbf{A}^i + M_{BB} \cdot \mathbf{B}^i, \end{cases}$$

which finally can be written as

$$\begin{pmatrix} \mathbf{A}^o \\ \mathbf{B}^o \end{pmatrix} = \begin{pmatrix} M_{AA} & M_{AB} \\ M_{BA} & M_{BB} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix} \equiv S \cdot \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix},$$

where we denote S as the Floquet flux matrix. Whenever the momentum is imaginary, in this case $n < 0$, there is an evanescent mode meaning it gets bound in the quantum well.

As a side note, without the presence of a static potential, $V_0 = 0$, Ref.[2] found that strong enough driving still induced quasi-bound states with a limited life-time, perhaps connected to Wigner delay.

Returning to the S -matrix, the $n < 0$ states are non-propagating evanescent modes (having imaginary momentum k_n), hence we discard them when calculating the total transmission. As it stands the S -matrix describe the Flux, making it a scattering matrix we must have unitarity. We obtain that by normalizing element-wise as

$$\begin{aligned}
s(E_n, E_m) &= \begin{pmatrix} s_{LL}(E_n, E_m) & s_{LR}(E_n, E_m) \\ s_{RL}(E_n, E_m) & s_{RR}(E_n, E_m) \end{pmatrix} \quad (20) \\
&= \sqrt{\frac{\text{Re}(k_n)}{\text{Re}(k_m)}} S_{nm} = \begin{pmatrix} r_{nm} & t'_{nm} \\ t_{nm} & r'_{nm} \end{pmatrix},
\end{aligned}$$

where we read $s_{\alpha\beta}(E_n, E_m)$ as the process where the particle enters from lead β with energy E_m and scatters into lead α with energy E_n . The indices can be either L (left) or R (right). We can thus identify the elements r_{nm} , t_{nm} as the amplitudes for reflecting (r) and transmitting (t) an incoming particle on the left from the m -th to the n -th sideband. Primed quantities correspond to scattering on the right side. For all calculations we assume a system where the incident particle approaches from the left in the zeroth Floquet sideband, thus carrying the momentum k_0 , with energy $E_0 = E_F$. The total transmission is

then calculated as

$$T = \sum_{n,m=-\infty}^{\infty} |s_{RL}(E_n, E_m)|^2 = \sum_{n=0}^{\infty} |t_{n0}|^2.$$

In order to avoid computationally dealing with infinitely large systems we truncate the number of sidebands to $n, m \in [-N, N]$, with a lower bound of $N \geq V_1/\hbar\omega$ to ensure stability demonstrated by Refs.[6, 7]. To elucidate the mechanism of exchange for the Fano peaks the total transmission, the zeroth to zeroth mode transmission, as well as the $t_{-1,0}$ flux before normalizing, are plotted over the Fermi energy. The results are displayed in Fig.[3]. The first observation is that a Fano resonance occurs at energy E_{Fano} , which is related to the first bound state energy by one quanta: $E_{Fano} - \hbar\omega = E_B$. For stronger driving and higher modes accounted for, more resonances at higher E_F emerge, noted in Ref.[2]. The middle plot illustrates how the major contribution lies within only the zeroth-mode transmission, when the electron just passes through. In the bottom plot a divergence arises at the resonant energy. This indicates a build-up of evanescent modes, or bound states. In Ref.[2] they attribute it to the incoming electron, when reaching the potential, interacting with the driving field such that it emits a quanta to the field at the point when it can de-excite into a bound state of the static potential. The electron then absorbs quanta, exciting it up into a propagating sideband, resulting in perfect transmission.

III. PUMPED SHOT-NOISE

The transmission spectrum can be captured in conductance measurements. Although, to better capture the transport for small numbers of quantized charge carriers, measuring shot-noise may provide significant signals. It is a measure of the current-current correlation across the scattering region. This technique is possible to use today due to the technological improvements in sensitive detection. We consider the pumped shot-noise, which is induced by the oscillating potential (our quantum pump), as was done in Refs.[3, 4] following the framework developed in Ref.[8]. The underlying formalism is restated

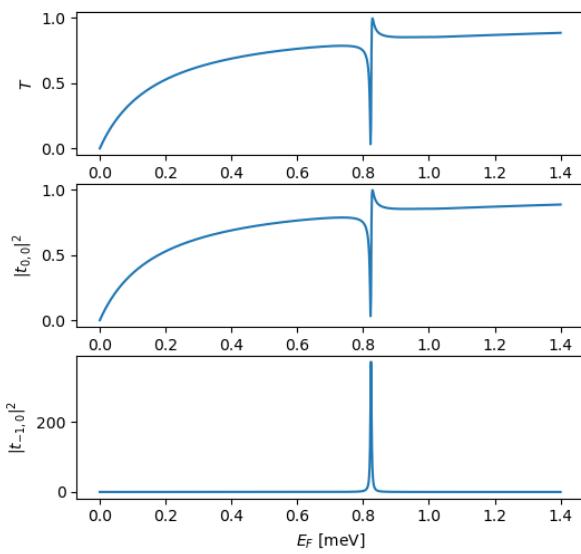


FIG. 3: Transmission of an incident electron in the zeroth Floquet sideband, through a time-varying potential well. (a) displays the total Transmission for $\sum_{n=0}^5 |t_{n0}|^2$. (b) shows the lowest sideband almost entirely captures the transmission behaviour. (c) shows a peak at the Floquet energy $E_F = E_b + \hbar\omega$ corresponding to the Fano resonances where the electron ejects a photon of energy $\hbar\omega$ dropping down into an evanescent mode.

here and further detailed to provide a clear step for implementing in numerical calculations. The same resulting expression is used for all subsequent systems studied. Zero -frequency, -temperature and bias is assumed for which the analytical expression is

$$\mathcal{N}_{\alpha\beta} = \frac{e}{2h^2} \int_0^\infty dE \sum_{\gamma\delta} \sum_{mnp \in \mathbb{Z}} M_{\alpha\beta\gamma\delta}(E, E_m, E_n, E_p) \times [f_0(E_n) - f_0(E_m)]^2 \quad (21)$$

where e and h are the fundamental charge and Planck's constant, respectively. The multi-variate element

$$M_{\alpha\beta\gamma\delta}(E, E_m, E_n, E_p) = s_{\alpha\gamma}^*(E, E_n) s_{\alpha\delta}(E, E_m) \times s_{\beta\delta}^*(E_p, E_m) s_{\beta\gamma}(E_p, E_n), \quad (22)$$

is the product of matrix elements described in Eq.(20) with indices $\alpha, \beta, \gamma, \delta \in \{L, R\}$ again denoting which lead is considered, where L, R denote the left and right lead respectively. The integrand described by Eq.[22] records the scattering process contributing to the pumped shot-noise, which reads as two particles entering from lead γ and δ with energies E_n and E_m , respectively. They are then scattered into leads α and β with energies E and E_p , respectively. Rephrased, the incoming pair are scattered from the n -th and m -th sideband, into the zeroth and p -th sidebands. The zero temperature Fermi-Dirac

distribution

$$f_0(E_n) = \frac{1}{e^{(E_n - E_F)/k_B T} + 1},$$

is used, and we define $\mathcal{F}_{nm} := [f_0(E_n) - f_0(E_m)]^2$ for easier notation. All the energies are taken as $E_j = E + j\hbar\omega$, with the same zeroth mode energy $E_0 = E$ in all quantities, corresponding to the integration parameter.

Pumped shot-noise is unitary, $\mathcal{N}_{LL} = -\mathcal{N}_{LR} = -\mathcal{N}_{RL} = \mathcal{N}_{RR}$, meaning we only need to consider one of the quantities which we choose to be $\mathcal{N}_{LL} =: \mathcal{N}$. We then have

$$\mathcal{N} = \frac{e}{2h^2} \int_0^\infty dE \sum_{\gamma\delta} \sum_{mnp \in \mathbb{Z}} M_{LL\gamma\delta}(E, E_m, E_n, E_p) \mathcal{F}_{nm}$$

Denoting , and noticing it is symmetric, we take a closer look at the integrand to arrive at a more explicit expression. We have

$$\begin{aligned} & \sum_{mnp \in \mathbb{Z}} M_{LL\gamma\delta}(E, E_m, E_n, E_p) \mathcal{F}_{nm} = \\ & \sum_{mnp \in \mathbb{Z}} s_{L\gamma}^*(E, E_n) s_{L\delta}(E, E_m) s_{L\delta}^*(E_p, E_m) \\ & \times s_{L\gamma}(E_p, E_n) \mathcal{F}_{mn} \end{aligned}$$

using the notation $s_{\alpha\beta}(E_i, E_j) =: (s_{\alpha\beta})_{ij}$ the expression above becomes

$$\begin{aligned} & \sum_{mnp \in \mathbb{Z}} (s_{L\gamma}^*)_{0n} (s_{L\delta})_{0m} (s_{L\delta}^*)_{pm} (s_{L\gamma})_{pn} \mathcal{F}_{mn} \\ & = \sum_{mn \in \mathbb{Z}} (s_{L\gamma}^\dagger)_{n0} (s_{L\delta})_{0m} (s_{L\delta}^\dagger s_{L\gamma})_{mn} \mathcal{F}_{mn} \\ & = \sum_{mn \in \mathbb{Z}} (s_{L\delta})_{0m} [(s_{L\delta}^\dagger s_{L\gamma}) \odot \mathcal{F}]_{mn} (s_{L\gamma}^\dagger)_{n0} \\ & = (s_{L\delta} [(s_{L\delta}^\dagger s_{L\gamma}) \odot \mathcal{F}] s_{L\gamma}^\dagger)_{00} \end{aligned}$$

where \odot is the Hadamard product, an element-wise product between two matrices. Including summation over leads we find the integrand to be

$$\begin{aligned} & \sum_{\gamma\delta} \sum_{mnp \in \mathbb{Z}} M_{LL\gamma\delta}(E, E_m, E_n, E_p) \mathcal{F}_{nm} \\ & = (r[[r^\dagger r] \odot \mathcal{F}] r^\dagger)_{00} + (r[[r^\dagger t']] \odot \mathcal{F}] t'^\dagger)_{00} \\ & \quad + (t'[[t'^\dagger r] \odot \mathcal{F}] r^\dagger)_{00} + (t'[[t'^\dagger t']] \odot \mathcal{F}] t'^\dagger)_{00} \\ & = (r[[r^\dagger r] \odot \mathcal{F}] r^\dagger + r[[r^\dagger t']] \odot \mathcal{F}] t'^\dagger \\ & \quad + t'[[t'^\dagger r] \odot \mathcal{F}] r^\dagger + t'[[t'^\dagger t']] \odot \mathcal{F}] t'^\dagger)_{00} \end{aligned}$$

For the computations we truncate the energy range to be $E \in [0, E_F]$, and only consider propagating modes which for our case are the non-negative values. With the previous lowest upper bound given by the oscillation strength we get $m, n, p \in [0, N]$, or $t', r \in \mathbb{C}^{(N+1) \times (N+1)}$

being a lower right block of s_{LR} and s_{LL} respectively. With these results the final expression we compute is

$$\mathcal{N} = \frac{e}{2\hbar^2} \sum_0^{E_F} \Delta E (r[[r^\dagger r] \odot \mathcal{F}] r^\dagger + r[[r^\dagger t'] \odot \mathcal{F}] t'^\dagger + t'[[t'^\dagger r] \odot \mathcal{F}] r^\dagger + t'[[t'^\dagger t'] \odot \mathcal{F}] t'^\dagger)_{00}$$

where ΔE is sum small energy step. Resonant scatterings has a suppressing or enhancing effect on the spectrum of \mathcal{N} . However, this effect might be too small to properly observe by inspection and therefore its differential $\frac{\partial \mathcal{N}}{\partial E_F}$ with respect to the Fermi energy is also calculated and plotted in the coming sections.

IV. 2DEG

The 2-dimensional electron gas in the xy -plane is the extension of the previous system to the 2D Hamiltonian

$$H = -\frac{\hbar^2}{2\mu} \nabla^2 + V(x, t)$$

Thus deriving the scattering matrix proceeds much the same except for a slight tweaking of wave-vectors. There is an added transversal dimension in y -direction which is invariant under the potential due to the potential well being considered infinitely extended in the y -direction. In real systems we consider the case of y -extension much larger than the potential width L . We make the wavefunction ansatz

$$\phi(x, y, t) = e^{ik_y y} \psi(x, t)$$

with $\psi(x, t)$ constructed the same as in the previous section. The resulting wave-vectors for regions I, III and II are

$$k_{xn}^2 = \frac{2\mu}{\hbar^2} E_n - k_y^2,$$

and

$$q_m^2 = \frac{2\mu}{\hbar^2} (E_m - V_0) - k_y^2,$$

respectively. Plugging these into the scattering matrix for the 1D case of section II we obtain the desired result. Increasing k_y is observed to shift the resonant transmission points to higher incoming energies, shown in the top plots of Fig.[4]. In Ref.[3] they observed that the resonant energies are now connected to the bound states according to

$$E_{Fano} - \hbar\omega - \frac{\hbar^2 k_y^2}{\mu} = E_b.$$

capturing the increasing energy requirement due to k_y . The result elucidates the previous matching condition of energies put forth by Ref.[2], to now identify it as a

wave-vector matching between the x -components of the evanescent mode and a bound state. In this setup it is $k_{x,-1} = k_{b,x}$.

The pumped shot-noise is shown in the middle plot of Fig.[4], displaying a monotone increase with higher energies followed by a monotone decrease. Before the peak there is a small kink for each graph, which is hard to notice. The differential shot-noise highlights these kinks as a resonance at the Fano energies.

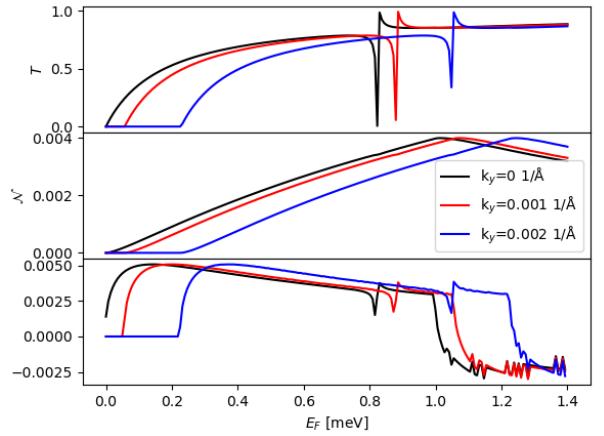


FIG. 4: Transmission for a 2-dimensional electron gas with an electrostatic potential well. Transversal invariance in the y -direction. Increased momentum in y -direction shifts resonant point towards higher energies. Pumped shot-noise is suppressed at resonant point. Its differential more clearly captures the effect.

V. GENERAL 2-LEVEL STATE

To reduce the work later on we diagonalise a general two-level system so that it later on can be handled through parameter mapping. The Hamiltonian for such a system can be expressed as

$$H = \alpha I + \beta \sigma_x + \gamma \sigma_y + \delta \sigma_z = \begin{pmatrix} \alpha + \delta & \beta - i\gamma \\ \beta + i\gamma & \alpha - \delta \end{pmatrix}$$

where α could correspond to some potential or tilt. We can identify the bloch-vector $\mathbf{r} = (\beta, \gamma, \delta)$ to write the Hamiltonian as

$$H = \alpha I + \mathbf{r} \cdot \boldsymbol{\sigma}$$

where $\boldsymbol{\sigma}$ is the Pauli vector. The time-independent part of the Schrödinger equation would then be

$$H\psi = E\psi$$

with E the energy-eigenvalues of the matrix H . Diagonalizing H we obtain the characteristic polynomial

$$\begin{aligned} 0 = \det(H - EI) &= \begin{vmatrix} \alpha + \delta - E & \beta - i\gamma \\ \beta + i\gamma & \alpha - \delta - E \end{vmatrix} \\ &= (\alpha - E)^2 - \delta^2 - (\beta^2 + \gamma^2) = (\alpha - E)^2 - |\mathbf{r}|^2. \end{aligned}$$

We let $r := |\mathbf{r}|$ for notational ease. Solving for E we obtain

$$E_{\pm} = \alpha \pm r$$

indicating the energy average is α with r splitting into two levels. The corresponding eigenstates are obtained as the parametric solutions to $N(H - E_{\pm}I)$, the nullspace. We get

$$\begin{aligned} N(H - E_{\pm}I) &= \begin{pmatrix} \alpha + \delta - E_{\pm} & \beta - i\gamma \\ \beta + i\gamma & \alpha - \delta - E_{\pm} \end{pmatrix} \\ &= \begin{pmatrix} \delta - (\pm r) & \beta - i\gamma \\ \beta + i\gamma & -\delta - (\pm r) \end{pmatrix} \\ &\sim \begin{pmatrix} \delta^2 - r^2 & (\beta - i\gamma)(\delta \pm r) \\ \beta^2 + \gamma^2 & -(\delta \pm r)(\beta - i\gamma) \end{pmatrix} \\ &= \begin{pmatrix} -(\beta^2 + \gamma^2) & (\beta - i\gamma)(\delta \pm r) \\ \beta^2 + \gamma^2 & -(\delta \pm r)(\beta - i\gamma) \end{pmatrix} \\ &\sim \begin{pmatrix} -(\beta^2 + \gamma^2) & (\beta - i\gamma)(\delta \pm r) \\ 0 & 0 \end{pmatrix} \\ &\sim \begin{pmatrix} -(\beta + i\gamma) & \delta \pm r \\ 0 & 0 \end{pmatrix} \end{aligned}$$

which is parametrized by

$$\chi_{\pm} = N_{\pm} \begin{pmatrix} 1 \\ \frac{\beta+i\gamma}{\delta \pm r} \end{pmatrix},$$

here we use N_{\pm} as the normalizing factor, so that the wf is in turn normalized. The explicit expression for N_{\pm} is obtained by solving

$$\begin{aligned} 1 &= \|\chi_{\pm}\| = N_{\pm} \sqrt{1 + \left(-\frac{\beta + i\gamma}{\delta \pm r}\right)^* \left(-\frac{\beta + i\gamma}{\delta \pm r}\right)} \\ \implies N_{\pm} &= \frac{1}{\sqrt{1 + \frac{\beta^2 + \gamma^2}{(\delta \pm r)^2}}} = \frac{r \pm \delta}{\sqrt{(r \pm \delta)^2 + \beta^2 + \gamma^2}} \\ &= \frac{r \pm \delta}{\sqrt{(r^2 \pm 2r\delta + \delta^2 + \beta^2 + \gamma^2)}} = \frac{r \pm \delta}{\sqrt{2r^2 \pm 2r\delta}} \\ &= \frac{r \pm \delta}{\sqrt{2r(r \pm \delta)}} = \sqrt{\frac{r \pm \delta}{2r}}. \end{aligned}$$

meaning the final form of the general eigenstates are

$$\chi = c_+ \sqrt{\frac{r + \delta}{2r}} \begin{pmatrix} 1 \\ -\frac{\beta+i\gamma}{r+\delta} \end{pmatrix} + c_- \sqrt{\frac{r - \delta}{2r}} \begin{pmatrix} 1 \\ \frac{\beta+i\gamma}{r-\delta} \end{pmatrix}.$$

Depending on the system, both vectors might have several attached phase factors such as plane waves, e.g.

$\exp(i\mathbf{k} \cdot \mathbf{x})$, determining the explicit expressions for the vector parameters. The coefficients c_{\pm} correspond to the energies E_{\pm} , meaning for E_+ (E_-) we keep c_+ (c_-). That is, when $E = \alpha + r$ ($E = \alpha - r$) rearranging we get $E - \alpha = r$ ($-[E - \alpha] = r$). Since r is a modulus $r \geq 0$ is always fulfilled, so we can write it as $E - \alpha \geq 0$ ($-[E - \alpha] \geq 0$). Finally this tells us we can write the coefficients in terms of the Heaviside function,

$$c_{\pm} = \Theta(\pm(E - \alpha)),$$

defined as

$$\Theta(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

The state is then

$$\begin{aligned} \chi &= \Theta(E - \alpha) \sqrt{\frac{r + \delta}{2r}} \begin{pmatrix} 1 \\ -\frac{\beta+i\gamma}{r+\delta} \end{pmatrix} \\ &\quad + \Theta(-[E - \alpha]) \sqrt{\frac{r - \delta}{2r}} \begin{pmatrix} 1 \\ \frac{\beta+i\gamma}{r-\delta} \end{pmatrix}. \end{aligned} \quad (23)$$

To clarify, in the case of $\alpha = \beta = 0 \neq \delta = r$ the procedure done above is not required, and furthermore could not be carried out the same due to division by zero. The Hamiltonian would already be diagonal.

VI. GRAPHENE

In graphene an electron-hole coupling form a massless quasiparticle, which close to the Dirac point display relativistic properties (linear dispersion). The Weyl-Dirac Hamiltonian is therefore employed,

$$i\hbar \frac{\partial}{\partial t} \Psi = (v_F \boldsymbol{\sigma} \cdot \mathbf{p} + V(x, t)I) \Psi, \quad (24)$$

where $v_F \approx c/300$ is the speed of the quasiparticle, a sort of effective speed of light. Considering the static case first we have

$$E\Psi(x, y) = (v_F \boldsymbol{\sigma} \cdot \mathbf{p} + V_0 I) \Psi(x, y), \quad (25)$$

where we can identify the system parameters as $\alpha = V_0$, $\beta = v_F p_x$, $\gamma = v_F p_y$, and $\delta = 0$ giving us $\mathbf{r} = (v_F p_x, v_F p_y, 0)$. We therefore have the energy

$$E_{\pm} = V_0 \pm v_F p,$$

where we keep the same expression for all regions but note that outside the potential well we put $V_0 = 0$. Assuming plane wave solutions we have $p_j = \hbar k_j$ and $\mathbf{r} = \hbar v_F \mathbf{k}$. We can then write

$$k = \pm \frac{E_{\pm} - V_0}{\hbar v_F} = \text{sign}(E - V_0) \frac{E - V_0}{\hbar v_F} \quad (26)$$

where the sign function is implemented to keep $k = |\mathbf{k}| \geq 0$. If we define $k_x = k \cos \phi$ and $k_y = k \sin \phi$ we get

$$\mathbf{r} = \hbar v_F k (\cos \phi, \sin \phi) = s \cdot (E - V_0) (\cos \phi, \sin \phi),$$

where we defined $s := \text{sign}(E - V_0)$. The normalizing factors are

$$N_{\pm} = \sqrt{\frac{r \pm \delta}{2r}} = \frac{1}{\sqrt{2}}$$

which we absorb into the amplitudes as it is constant and won't affect the overall dynamics. For the spinor elements, we have the case of $\beta = \pm \hbar v_F k_x$ for the corresponding plane waves $e^{\pm i k_x x}$ so that

$$\frac{\beta + i\gamma}{r} = \frac{s \cdot (E - V_0)(\pm \cos \phi + i \sin \phi)}{(E - V_0)} = -se^{\pm i\phi},$$

where $\phi = \arctan(k_y/k_x)$. Plugging all of this into Eq.(23), for $\Psi(x, y) = e^{ik_y y} \psi(x) = e^{ik_y y} (\psi_1(x), \psi_2(x))^T$

we obtain

$$\begin{aligned} \psi = & \Theta(E - V_0) [a \begin{pmatrix} 1 \\ se^{i\phi} \end{pmatrix} e^{ik_x x} + b \begin{pmatrix} 1 \\ -se^{-i\phi} \end{pmatrix} e^{-ik_x x}] \\ & + \Theta(-[E - V_0]) [a \begin{pmatrix} 1 \\ -se^{i\phi} \end{pmatrix} e^{ik_x x} + b \begin{pmatrix} 1 \\ se^{-i\phi} \end{pmatrix} e^{-ik_x x}]. \end{aligned}$$

Notice that $s\Theta(-[E - V_0]) = -s\Theta(E - V_0)$, meaning the two rows of the solution are equivalent, we keep the first:

$$\begin{cases} \psi_1(x) = ae^{ik_x x} + be^{-ik_x x}, \\ \psi_2(x) = ase^{ik_x x + i\phi} - bse^{-ik_x x - i\phi}. \end{cases}$$

Including the quantum pump Floquet's theorem as in previous sections, expanding for all possible modes we obtain the total wavefunction

$$\psi_n(x, t) = e^{-iE_n t/\hbar} \begin{cases} A_n^i \begin{pmatrix} 1 \\ s_n e^{i\phi_n} \end{pmatrix} e^{ik_{xn} x} - A_n^o \begin{pmatrix} -1 \\ s_n e^{-i\phi_n} \end{pmatrix} e^{-ik_{xn} x}, \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[a_m \begin{pmatrix} 1 \\ s'_m e^{i\theta_m} \end{pmatrix} e^{iq_m x} - b_m \begin{pmatrix} -1 \\ s'_m e^{-i\theta_m} \end{pmatrix} e^{-iq_m x} \right], \\ -B_n^i \begin{pmatrix} -1 \\ s_n e^{-i\phi_n} \end{pmatrix} e^{-ik_{xn} x} + B_n^o \begin{pmatrix} 1 \\ s_n e^{i\phi_n} \end{pmatrix} e^{ik_{xn} x}, \end{cases} \quad (27)$$

where we have defined $J_{nm} := J_{n-m}(V_1/\hbar\omega)$ and use the indexed notation $s_n = \text{sign}(E_n)$, $s'_m = \text{sign}(E_m - V_0)$. The momenta, solved from Eq.(26), are

$$k_{xn}^2 = \frac{E_n^2}{(\hbar v_F k_y)^2} - k_y^2, \quad q_m^2 = \frac{(E_m - V_0)^2}{(\hbar v_F k_y)^2} - k_y^2. \quad (28)$$

For the S-matrix derivation we need only consider the wf continuity at the boundaries, because we expect the spinor to be continuous elementwise it provides enough equations to solve. The relations are

$$\begin{aligned} \psi_{1,n}^I(-L/2, t) &= \psi_{1,n}^{II}(-L/2, t), \\ \psi_{2,n}^I(-L/2, t) &= \psi_{2,n}^{II}(-L/2, t), \\ \psi_{1,n}^{III}(L/2, t) &= \psi_{1,n}^{II}(L/2, t), \\ \psi_{2,n}^{III}(L/2, t) &= \psi_{2,n}^{II}(L/2, t), \end{aligned} \quad (29)$$

explicitly expressed as

$$\begin{aligned} A_n^i e^{-ik_{xn} L/2} + A_n^o e^{ik_{xn} L/2} = & \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[a_m e^{-iq_m L/2} + b_m e^{iq_m L/2} \right], & \end{aligned} \quad (30)$$

$$\begin{aligned} A_n^i s_n e^{-ik_{xn} L/2 + i\phi_n} - A_n^o s_n e^{ik_{xn} L/2 - i\phi_n} = & \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[s'_m a_m e^{-iq_m L/2 + i\theta_m} - s'_m b_m e^{iq_m L/2 - i\theta_m} \right], & \end{aligned} \quad (31)$$

$$\begin{aligned} B_n^i e^{-ik_{xn} L/2} + B_n^o e^{ik_{xn} L/2} = & \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[a_m e^{iq_m L/2} + b_m e^{-iq_m L/2} \right], & \end{aligned} \quad (32)$$

and

$$\begin{aligned} -B_n^i s_n e^{-ik_{xn} L/2 - i\phi_n} + B_n^o s_n e^{ik_{xn} L/2 + i\phi_n} = & \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[s'_m a_m e^{iq_m L/2 + i\theta_m} - s'_m b_m e^{-iq_m L/2 - i\theta_m} \right], & \end{aligned} \quad (33)$$

respectively. We compute $s_n e^{-i\phi_n} (30) + (31)$ and $s_n e^{i\phi_n} (32) - (33)$ to get

$$\begin{aligned} 2 \cos(\phi_n) s_n e^{-ik_{xn}L/2} A_n^i = \\ \sum_{m=-\infty}^{\infty} J_{nm} [a_m e^{-iq_m L/2} (s_n e^{-i\phi_n} + s'_m e^{i\theta_m}) \\ + b_m e^{iq_m L/2} (s_n e^{-i\phi_n} - s'_m e^{-i\theta_m})], \end{aligned} \quad (34)$$

and

$$\begin{aligned} 2 \cos(\phi_n) s_n e^{-ik_{xn}L/2} B_n^i = \\ \sum_{m=-\infty}^{\infty} J_{nm} [a_m e^{iq_m L/2} (s_n e^{i\phi_n} - s'_m e^{i\theta_m}) \\ + b_m e^{-iq_m L/2} (s_n e^{i\phi_n} + s'_m e^{-i\theta_m})], \end{aligned} \quad (35)$$

respectively. Finally we compute $(34) \pm (35)$:

$$\begin{aligned} 2 \cos(\phi_n) s_n e^{-ik_{xn}L/2} (A_n^i \pm B_n^i) = \\ \sum_{m=-\infty}^{\infty} J_{nm} \{ a_m [e^{-iq_m L/2} (s_n e^{-i\phi_n} + s'_m e^{i\theta_m}) \\ \pm e^{iq_m L/2} (s_n e^{i\phi_n} - s'_m e^{i\theta_m})] \\ + b_m [e^{iq_m L/2} (s_n e^{-i\phi_n} - s'_m e^{-i\theta_m}) \\ \pm e^{-iq_m L/2} (s_n e^{i\phi_n} + s'_m e^{-i\theta_m})] \}. \end{aligned} \quad (36)$$

Defining the matrices

$$\begin{aligned} (\mathbf{M}_{sa}^{\pm})_{nm} := & [e^{-iq_m L/2} (s_n e^{-i\phi_n} + s'_m e^{i\theta_m}) \\ & \pm e^{iq_m L/2} (s_n e^{i\phi_n} - s'_m e^{i\theta_m})] J_{nm}, \\ (\mathbf{M}_{sb}^{\pm})_{nm} := & [e^{iq_m L/2} (s_n e^{-i\phi_n} - s'_m e^{-i\theta_m}) \\ & \pm e^{-iq_m L/2} (s_n e^{i\phi_n} + s'_m e^{-i\theta_m})] J_{nm}, \\ (\mathbf{M}_r)_{nm} := & 2 \cos(\phi_n) s_n e^{-ik_{xn}L/2} \delta_{nm}, \\ (\mathbf{M}_i)_{nm} := & e^{-ik_{xn}L} \delta_{nm}, \\ (\mathbf{M}_c^{\pm})_{nm} := & e^{-i(k_{xn} \pm q_m)L/2} J_{nm}, \end{aligned}$$

we reformulate Eq.(36) as

$$(\mathbf{M}_r)_{nm} (A_n^i \pm B_n^i) = \sum_{m=-\infty}^{\infty} (\mathbf{M}_{sa}^{\pm})_{nm} a_m + (\mathbf{M}_{sb}^{\pm})_{nm} b_m$$

or in the more compact form

$$\mathbf{M}_r (\mathbf{A}^i \pm \mathbf{B}^i) = \mathbf{M}_{sa}^{\pm} \mathbf{a} + \mathbf{M}_{sb}^{\pm} \mathbf{b}. \quad (37)$$

Again we proceed to isolate \mathbf{a} and \mathbf{b} using Eq.(37),

$$\begin{aligned} \left\{ \begin{array}{l} \mathbf{M}_r (\mathbf{A}^i + \mathbf{B}^i) = \mathbf{M}_{sa}^+ \mathbf{a} + \mathbf{M}_{sb}^+ \mathbf{b} \\ \mathbf{M}_r (\mathbf{A}^i - \mathbf{B}^i) = \mathbf{M}_{sa}^- \mathbf{a} + \mathbf{M}_{sb}^- \mathbf{b} \end{array} \right. \\ \implies \left\{ \begin{array}{l} (\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_r (\mathbf{A}^i + \mathbf{B}^i) = (\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_{sa}^+ \mathbf{a} + \mathbf{b} \\ (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_r (\mathbf{A}^i - \mathbf{B}^i) = (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_{sa}^- \mathbf{a} + \mathbf{b} \end{array} \right. \end{aligned}$$

eliminating \mathbf{b} gives

$$\begin{aligned} (\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_r (\mathbf{A}^i + \mathbf{B}^i) - (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_r (\mathbf{A}^i - \mathbf{B}^i) \\ = \left[(\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_{sa}^+ - (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_{sa}^- \right] \mathbf{a} \\ \implies \left[(\mathbf{M}_{sb}^+)^{-1} - (\mathbf{M}_{sb}^-)^{-1} \right] \mathbf{M}_r \mathbf{A}^i \\ + \left[(\mathbf{M}_{sb}^+)^{-1} + (\mathbf{M}_{sb}^-)^{-1} \right] \mathbf{M}_r \mathbf{B}^i \\ = \left[(\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_{sa}^+ - (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_{sa}^- \right] \mathbf{a} \end{aligned}$$

which we write as

$$\mathbf{a} = \mathbf{a}_A \mathbf{A}^i + \mathbf{a}_B \mathbf{B}^i, \quad (38)$$

where we have defined

$$\begin{aligned} \mathbf{a}_A := & \left[(\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_{sa}^+ - (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_{sa}^- \right]^{-1} \\ & \times \left[(\mathbf{M}_{sb}^+)^{-1} - (\mathbf{M}_{sb}^-)^{-1} \right] \mathbf{M}_r, \\ \mathbf{a}_B := & \left[(\mathbf{M}_{sb}^+)^{-1} \mathbf{M}_{sa}^+ - (\mathbf{M}_{sb}^-)^{-1} \mathbf{M}_{sa}^- \right]^{-1} \\ & \times \left[(\mathbf{M}_{sb}^+)^{-1} + (\mathbf{M}_{sb}^-)^{-1} \right] \mathbf{M}_r. \end{aligned}$$

Similarly we can obtain

$$\mathbf{b} = \mathbf{b}_A \mathbf{A}^i + \mathbf{b}_B \mathbf{B}^i, \quad (39)$$

with

$$\begin{aligned} \mathbf{b}_A := & \left[(\mathbf{M}_{sa}^+)^{-1} \mathbf{M}_{sb}^+ - (\mathbf{M}_{sa}^-)^{-1} \mathbf{M}_{sb}^- \right]^{-1} \\ & \times \left[(\mathbf{M}_{sa}^+)^{-1} - (\mathbf{M}_{sa}^-)^{-1} \right] \mathbf{M}_r, \\ \mathbf{b}_B := & \left[(\mathbf{M}_{sa}^+)^{-1} \mathbf{M}_{sb}^+ - (\mathbf{M}_{sa}^-)^{-1} \mathbf{M}_{sb}^- \right]^{-1} \\ & \times \left[(\mathbf{M}_{sa}^+)^{-1} + (\mathbf{M}_{sa}^-)^{-1} \right] \mathbf{M}_r. \end{aligned}$$

We isolate A_n^o and B_n^o from Eqs.(30, 32) written in the matrix notation as

$$\begin{cases} \mathbf{A}^o = \mathbf{M}_c^+ \mathbf{a} + \mathbf{M}_c^- \mathbf{b} - \mathbf{M}_i \mathbf{A}^i \\ \mathbf{B}^o = \mathbf{M}_c^- \mathbf{a} + \mathbf{M}_c^+ \mathbf{b} - \mathbf{M}_i \mathbf{B}^i. \end{cases}$$

Applying the notation from Eqs.(38, 39) gives

$$\begin{cases} \mathbf{A}^o = & \mathbf{M}_c^+ (\mathbf{a}_A \mathbf{A}^i + \mathbf{a}_B \mathbf{B}^i) \\ & + \mathbf{M}_c^- (\mathbf{b}_A \mathbf{A}^i + \mathbf{b}_B \mathbf{B}^i) - \mathbf{M}_i \mathbf{A}^i \\ \mathbf{B}^o = & \mathbf{M}_c^- (\mathbf{a}_A \mathbf{A}^i + \mathbf{a}_B \mathbf{B}^i) \\ & + \mathbf{M}_c^+ (\mathbf{b}_A \mathbf{A}^i + \mathbf{b}_B \mathbf{B}^i) - \mathbf{M}_i \mathbf{B}^i \end{cases}$$

or

$$\begin{cases} \mathbf{A}^o = & (\mathbf{M}_c^+ \mathbf{a}_A + \mathbf{M}_c^- \mathbf{b}_A - \mathbf{M}_i) \mathbf{A}^i \\ & + (\mathbf{M}_c^+ \mathbf{a}_B + \mathbf{M}_c^- \mathbf{b}_B) \mathbf{B}^i, \\ \mathbf{B}^o = & (\mathbf{M}_c^- \mathbf{a}_A + \mathbf{M}_c^+ \mathbf{b}_A) \mathbf{A}^i \\ & + (\mathbf{M}_c^- \mathbf{a}_B + \mathbf{M}_c^+ \mathbf{b}_B - \mathbf{M}_i) \mathbf{B}^i. \end{cases} \quad (40)$$

Defining

$$\begin{aligned} \mathbf{M}_{\mathbf{AA}} &= \mathbf{M}_c^+ \mathbf{a}_A + \mathbf{M}_c^- \mathbf{b}_A - \mathbf{M}_i \\ \mathbf{M}_{\mathbf{AB}} &= \mathbf{M}_c^+ \mathbf{a}_B + \mathbf{M}_c^- \mathbf{b}_B \\ \mathbf{M}_{\mathbf{BA}} &= \mathbf{M}_c^- \mathbf{a}_A + \mathbf{M}_c^+ \mathbf{b}_A \\ \mathbf{M}_{\mathbf{BB}} &= \mathbf{M}_c^- \mathbf{a}_B + \mathbf{M}_c^+ \mathbf{b}_B - \mathbf{M}_i \end{aligned}$$

we can write Eq.(40) in the familiar form

$$\begin{pmatrix} \mathbf{A}^o \\ \mathbf{B}^o \end{pmatrix} = \begin{pmatrix} \mathbf{M}_{\mathbf{AA}} & \mathbf{M}_{\mathbf{AB}} \\ \mathbf{M}_{\mathbf{BA}} & \mathbf{M}_{\mathbf{BB}} \end{pmatrix} \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix} = \mathbf{S} \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix}$$

where we once again have the S-matrix.

The transmission spectra is generated for different parameters than the electron-gas cases. We can still observe three distinct resonances with a slight difference in k_y , captured in the differential shot-noise, depicted in Fig.[5]. The shot-noise is even more non-descriptive for Graphene, motivating the use of the differential further. Resonances are found at much higher energies than in 2DEG, requiring a substantial increase in computation for the shot-noise calculations. The jaggedness in the differential plot is a computational artefact due to time limitations. The shape for the different k_y are different, sharpening the Fano peaks for higher momentum. Interestingly they are shifted in the opposite direction to previous results, higher y -momentum lowers the value of E_{Fano} . Due to higher driving frequency the number of modes are truncated to $N = 2$.

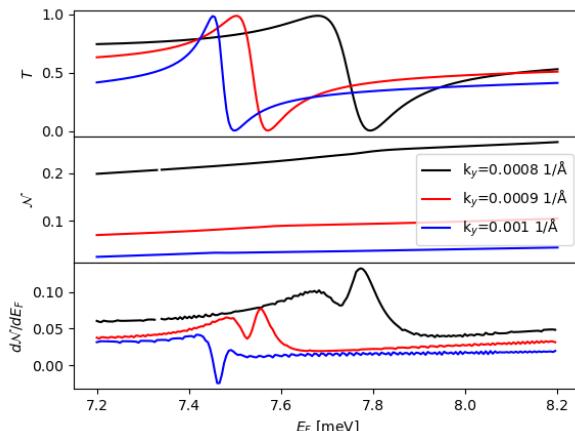


FIG. 5: Transmission and shot noise of graphene. Increased momentum now lowers the resonant energy.

VII. QBT

We assume the Hamiltonian used in [4], representing a system with a Brillouin zone of C_6 rotational symmetry, also harbouring a point of quadratic band touching,

$$H = \frac{\hbar^2}{2\mu}(2k_x k_y \sigma_x + (k_y^2 - k_x^2)\sigma_z) + V(x, t)I. \quad (41)$$

where we identify

$$\begin{aligned} \alpha &= V_0, \quad \beta = \frac{\hbar^2}{2\mu} 2k_x k_y, \quad \gamma = 0, \\ \delta &= \frac{\hbar^2}{2\mu} (k_y^2 - k_x^2), \quad r = \frac{\hbar^2}{2\mu} (k_x^2 + k_y^2), \\ E_{\pm} &= V_0 \pm \frac{\hbar^2}{2\mu} (k_x^2 + k_y^2). \end{aligned}$$

We solve for the momentum to get

$$k_x^{\pm} = \pm \sqrt{\frac{2\mu}{\hbar^2} |E - V_0| - k_y^2},$$

where we have considered $\pm(E_{\pm} - V_0) = |E - V_0|$. The corresponding wf, if we pick $k_x = k_x^+$, is

$$\begin{aligned} \psi = & \frac{|k_y|}{\sqrt{k_x^2 + k_y^2}} \left[\left(\frac{1}{k_x} \right) e^{ik_x x} + \left(-\frac{1}{k_y} \right) e^{-ik_x x} \right] \Theta(E - V_0) \\ & + \frac{|k_x|}{\sqrt{k_x^2 + k_y^2}} \left[\left(-\frac{1}{k_y} \right) e^{ik_x x} + \left(\frac{1}{k_x} \right) e^{-ik_x x} \right] \Theta(-[E - V_0]). \end{aligned}$$

For the incident or outgoing electron we only consider the conduction band for transport, meaning the second row of the wf vanishes for the outer regions. By considering the dynamic potential and the Floquet formalism we obtain the total wf

$$\psi_n(x, y, t) = e^{ik_y y} \begin{cases} \eta_{1,n} \left(A_n^i \left(\frac{1}{k_y} \right) e^{ik_{xn}x} + A_n^o \left(-\frac{1}{k_y} \right) e^{-ik_{xn}x} \right) \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[\eta_{2,m} \left(a_m \left(\frac{1}{q_m} \right) e^{iq_m x} + b_m \left(-\frac{1}{q_m} \right) e^{-iq_m x} \right) \Theta(E_m - V_0) \right. \\ \quad \left. + \eta_{3,m} \left(a_m \left(-\frac{1}{q_m} \right) e^{iq_m x} + b_m \left(\frac{1}{q_m} \right) e^{-iq_m x} \right) \Theta(-[E_m - V_0]) \right] \\ \eta_{1,n} \left(B_n^o \left(\frac{1}{k_xn} \right) e^{ik_{xn}x} + B_n^i \left(-\frac{1}{k_xn} \right) e^{-ik_{xn}x} \right) \end{cases} \quad (42)$$

where we have defined the normalizing factors

$$\begin{aligned} \eta_{1,n} &= \frac{|k_y|}{\sqrt{k_{xn}^2 + k_y^2}}, \quad \eta_{2,m} = \frac{|k_y|}{\sqrt{q_m^2 + k_y^2}}, \\ \eta_{3,m} &= \frac{|q_m|}{\sqrt{q_m^2 + k_y^2}}, \end{aligned}$$

The boundary conditions are the same as for the Graphene, and the procedure is performed similarly. To decrease clutter we define $\Theta_m^\pm := \Theta(\pm[E_m - \alpha])$. We find

$$\begin{aligned} \eta_{1,n} \left(A_n^i e^{-ik_{xn}L/2} + A_n^o e^{ik_{xn}L/2} \right) &= \\ \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} &\left(a_m e^{-iq_m L/2} + b_m e^{iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} &\left(a_m e^{-iq_m L/2} + b_m e^{iq_m L/2} \right) \Theta_m^-], \end{aligned} \quad (43)$$

$$\begin{aligned} \eta_{1,n} \frac{k_{xn}}{k_y} \left(A_n^i e^{-ik_{xn}L/2} - A_n^o e^{ik_{xn}L/2} \right) &= \\ \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} \frac{q_m}{k_y} &\left(a_m e^{-iq_m L/2} - b_m e^{iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \frac{k_y}{q_m} &\left(-a_m e^{-iq_m L/2} + b_m e^{iq_m L/2} \right) \Theta_m^-], \end{aligned} \quad (44)$$

$$\begin{aligned} \eta_{1,n} \left(B_n^o e^{ik_{xn}L/2} + B_n^i e^{-ik_{xn}L/2} \right) &= \\ \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} &\left(a_m e^{iq_m L/2} + b_m e^{-iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} &\left(a_m e^{iq_m L/2} + b_m e^{-iq_m L/2} \right) \Theta_m^-], \end{aligned} \quad (45)$$

and

$$\begin{aligned} \eta_{1,n} \frac{k_{xn}}{k_y} \left(B_n^o e^{ik_{xn}L/2} - B_n^i e^{-ik_{xn}L/2} \right) &= \\ \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} \frac{q_m}{k_y} &\left(a_m e^{iq_m L/2} - b_m e^{-iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \frac{k_y}{q_m} &\left(-a_m e^{iq_m L/2} + b_m e^{-iq_m L/2} \right) \Theta_m^-], \end{aligned} \quad (46)$$

for the first and second element of the spinors in regards to the left and right side respectively. We compute (43) + k_y/k_{xn} (44) and (45) - k_y/k_{xn} (46) to obtain

$$2\eta_{1,n}A_n^i e^{-ik_{xn}L/2} = \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} \left(a_m (1 + \frac{q_m}{k_{xn}}) e^{-iq_m L/2} + b_m (1 - \frac{q_m}{k_{xn}}) e^{iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \left(a_m (1 - \frac{k_y^2}{k_{xn} q_m}) e^{-iq_m L/2} + b_m (1 + \frac{k_y^2}{k_{xn} q_m}) e^{iq_m L/2} \right) \Theta_m^-], \quad (47)$$

$$2\eta_{1,n}B_n^i e^{-ik_{xn}L/2} = \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} \left(a_m (1 - \frac{q_m}{k_{xn}}) e^{iq_m L/2} + b_m (1 + \frac{q_m}{k_{xn}}) e^{-iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \left(a_m (1 + \frac{k_y^2}{k_{xn} q_m}) e^{iq_m L/2} + b_m (1 - \frac{k_y^2}{k_{xn} q_m}) e^{-iq_m L/2} \right) \Theta_m^-], \quad (48)$$

respectively. Combining (47) \pm (48) yields

$$2\eta_{1,n}e^{-ik_{xn}L/2}(A_n^i \pm B_n^i) = \sum_{m=-\infty}^{\infty} J_{nm} [\eta_{2,m} \left((a_m \pm b_m) (1 + \frac{q_m}{k_{xn}}) e^{-iq_m L/2} \pm (a_m \pm b_m) (1 - \frac{q_m}{k_{xn}}) e^{iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \left((a_m \pm b_m) (1 - \frac{k_y^2}{k_{xn} q_m}) e^{-iq_m L/2} \pm (a_m \pm b_m) (1 + \frac{k_y^2}{k_{xn} q_m}) e^{iq_m L/2} \right) \Theta_m^-]. \quad (49)$$

If we define $C_m^\pm := a_m \pm b_m$ we can simplify Eq.(49) as

$$2\eta_{1,n}e^{-ik_{xn}L/2}(A_n^i \pm B_n^i) = \sum_{m=-\infty}^{\infty} J_{nm} C_m^\pm [\eta_{2,m} \left((1 + \frac{q_m}{k_{xn}}) e^{-iq_m L/2} \pm (1 - \frac{q_m}{k_{xn}}) e^{iq_m L/2} \right) \Theta_m^+ \\ + \eta_{3,m} \left((1 - \frac{k_y^2}{k_{xn} q_m}) e^{-iq_m L/2} \pm (1 + \frac{k_y^2}{k_{xn} q_m}) e^{iq_m L/2} \right) \Theta_m^-]. \quad (50)$$

Defining the matrix elements

$$(M_r)_{nm} := 2\eta_{1,n}e^{-ik_{xn}L/2}\delta_{nm}, \\ (M_s^\pm)_{nm} := J_{nm} [\eta_{2,m} \left((1 + \frac{q_m}{k_{xn}}) e^{-iq_m L/2} \pm (1 - \frac{q_m}{k_{xn}}) e^{iq_m L/2} \right) \Theta_m^+, \\ + \eta_{3,m} \left((1 - \frac{k_y^2}{k_{xn} q_m}) e^{-iq_m L/2} \pm (1 + \frac{k_y^2}{k_{xn} q_m}) e^{iq_m L/2} \right) \Theta_m^-], \\ (M_c^\pm)_{nm} = \frac{J_{nm}}{\eta_{1,n}} (\eta_{2,m} \Theta(\varepsilon_m) + \eta_{3,m} \Theta(-\varepsilon_m)) e^{-i(k_{xn} \pm q_m)L/2}, \\ (M_i)_{nm} = e^{-ik_{xn}L} \delta_{nm}, \quad (51)$$

lets us write Eq.(50) as

$$\mathbf{M}_r \cdot (\mathbf{A}^i \pm \mathbf{B}^i) = \mathbf{M}_s^\pm \cdot \mathbf{C}^\pm$$

meaning we can isolate C as

$$\mathbf{C}^\pm = (\mathbf{M}_s^\pm)^{-1} \cdot \mathbf{M}_r \cdot (\mathbf{A}^i \pm \mathbf{B}^i)$$

from which we get

$$\mathbf{a} = \frac{1}{2}(\mathbf{C}^+ + \mathbf{C}^-), \quad \mathbf{b} = \frac{1}{2}(\mathbf{C}^+ - \mathbf{C}^-).$$

We will use these in (45) and (47) to isolate the outgoing quantities. First we define the matrix elements so that

we can write the outgoing as

$$\mathbf{A}^o = \mathbf{M}_c^+ \cdot \mathbf{a} + \mathbf{M}_c^- \cdot \mathbf{b} - \mathbf{M}_i \cdot \mathbf{A}^i, \\ \mathbf{B}^o = \mathbf{M}_c^- \cdot \mathbf{a} + \mathbf{M}_c^+ \cdot \mathbf{b} - \mathbf{M}_i \cdot \mathbf{A}^i. \quad (52)$$

Using the same notation as in Eqs.(38,39) but with

$$\mathbf{a}_A := \frac{1}{2}((\mathbf{M}_s^+)^{-1} + (\mathbf{M}_s^-)^{-1}) \cdot \mathbf{M}_r, \\ \mathbf{a}_B := \frac{1}{2}((\mathbf{M}_s^+)^{-1} - (\mathbf{M}_s^-)^{-1}) \cdot \mathbf{M}_r, \\ \mathbf{b}_A := \mathbf{a}_B, \quad \mathbf{b}_B := \mathbf{a}_A,$$

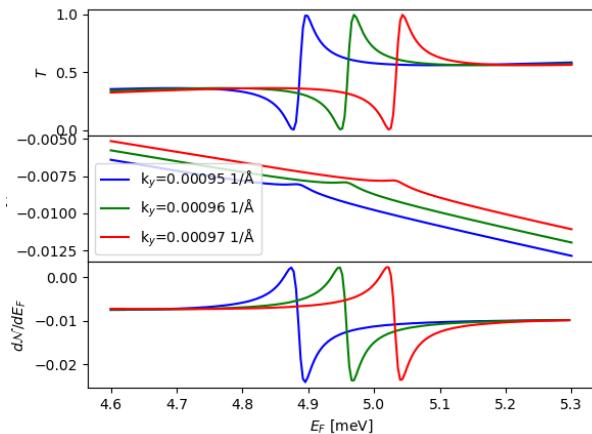


FIG. 6: Transmission, shot-noise, and the differential shot-noise, respectively of a quasiparticle in a generic QBT material. Fano peaks are shifted to higher energies with increased k_y

we can write Eq.(52) as

$$\begin{aligned} \mathbf{A}^o &= (\mathbf{M}_c^+ \cdot \mathbf{a}_A + \mathbf{M}_c^- \cdot \mathbf{b}_A - \mathbf{M}_i) \cdot \mathbf{A}^i \\ &\quad + (\mathbf{M}_c^+ \mathbf{a}_B + \mathbf{M}_c^- \mathbf{b}_B) \cdot \mathbf{B}^i \\ \mathbf{B}^o &= (\mathbf{M}_c^- \cdot \mathbf{a}_A + \mathbf{M}_c^+ \cdot \mathbf{b}_A) \cdot \mathbf{A}^i \\ &\quad + (\mathbf{M}_c^- \cdot \mathbf{a}_B + \mathbf{M}_c^+ \cdot \mathbf{b}_B - \mathbf{M}_i) \cdot \mathbf{B}^i, \end{aligned}$$

and finally put in the familiar S-matrix form

$$\begin{pmatrix} \mathbf{A}^o \\ \mathbf{B}^o \end{pmatrix} = \begin{pmatrix} \mathbf{M}_{AA} & \mathbf{M}_{AB} \\ \mathbf{M}_{BA} & \mathbf{M}_{BB} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix} \equiv \mathbf{S} \cdot \begin{pmatrix} \mathbf{A}^i \\ \mathbf{B}^i \end{pmatrix}. \quad (53)$$

In Fig.[6] three values of k_y are again used for plotting the transmission, shot-noise, and the differential of the shot-noise. Similar system parameters to the graphene in Fig.[5] were used, but the qualitative signatures are more akin to those of the 2DEG of Fig.[4]. This seems to be related to whether the model used is of linear or quadratic dispersion. For the energy range used the shift of Fano peaks are again to higher energies with increased momentum. However, in Ref.[4] they observe the shifts to alter this characteristic depending on over what range of momentum the spectra are computed, thus connected to the propagation angle to the x -axis. Furthermore they categorise whether a peak first drops to zero then spikes up to one, or vice versa, as two distinct types of Fano peaks.

VIII. QBT WITH TILT

We now aim to augment the previous sections Hamiltonian with a linear tilting term. They are named such due to the effect they have on the band structure. The

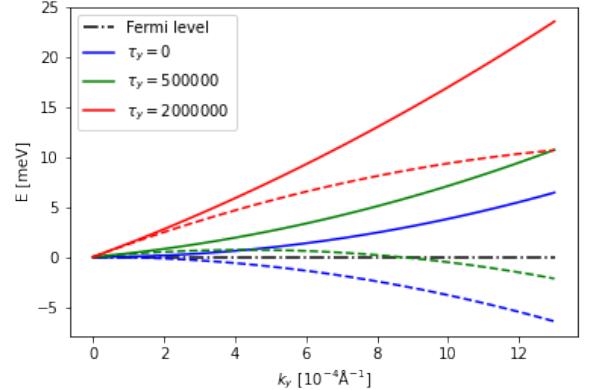


FIG. 7: Band structure for $k_x = 0$. Conduction (valence) bands are solid (dashed). For non-zero tilt there is a Weyl-semimetal type transition from I to II.

band topology is tilted in the direction of the tilting term. Introducing $\tau_y p_y$ to the Hamiltonian it becomes

$$H = \frac{\hbar^2}{2\mu} (2k_x k_y \sigma_x + (k_y^2 - k_x^2) \sigma_z) + V(x, t) + \tau_y \hbar k_y$$

from which we can identify the only change occurring for the parameter

$$\alpha = \tau_y \hbar k_y + V_0,$$

from which we get the new energy as

$$E_{\pm} = \tau_y \hbar k_y + V_0 \pm \frac{\hbar^2}{2\mu} (k_x^2 + k_y^2).$$

Viewing a k_x resolved band plot would raise (lower) the bands if the tilt τ_y and momentum k_y have the same (different) signs. If instead we regard the bands from a k_y resolved perspective, the system will transition from a type I to a type II Weyl-semimetal for any non-zero tilting, depicted in Fig.[7].

Solving for the momentum we obtain

$$k_x^{\pm} = \pm \sqrt{\frac{2\mu}{\hbar^2} |E - V_0 - \hbar \tau_y k_y| - k_y^2}$$

which we can observe to be similar to the previous section with just an extra term inside the absolute value. We can then apply similar strategies, very much like we did for the 2DEG compared to the 1D case. We will have the same derivations and expressions, with only plugging in these new momenta and adjusting the attached Heaviside functions. Again, taking $k_x = k_x^+$ the wf is

$$\begin{aligned} \psi &= \left[\frac{|k_y|}{\sqrt{k_1^2 + k_y^2}} \left(\frac{1}{k_x} \right) e^{ik_x x} + \frac{|k_y|}{\sqrt{k_x^2 + k_y^2}} \left(-\frac{1}{k_x} \right) e^{-ik_x x} \right] \Theta^+ \\ &\quad + \left[\frac{|k_x|}{\sqrt{k_x^2 + k_y^2}} \left(\frac{1}{k_y} \right) e^{ik_x x} + \frac{|k_x|}{\sqrt{k_x^2 + k_y^2}} \left(\frac{1}{k_y} \right) e^{-ik_x x} \right] \Theta^- \end{aligned}$$

We consider the conduction band for outer regions as in the previous section. We therefore keep only Θ^+ for

regions I and III. Introducing the pump we get the total wf as

$$\psi_n(x, y, t) = e^{ik_y y} \begin{cases} \eta_{1,n} \left(A_n^i \left(\frac{1}{k_x n} \right) e^{ik_x n x} + A_n^o \left(-\frac{1}{k_x n} \right) e^{-ik_x n x} \right) \\ \sum_{m=-\infty}^{\infty} J_{nm} \left[\eta_{2,m} \left(a_m \left(\frac{q_m}{k_y} \right) e^{iq_m x} + b_m \left(-\frac{q_m}{k_y} \right) e^{-iq_m x} \right) \Theta(E_m - V_0) \right. \\ \left. + \eta_{3,m} \left(a_m \left(-\frac{1}{k_y q_m} \right) e^{iq_m x} + b_m \left(\frac{1}{q_m k_y} \right) e^{-iq_m x} \right) \Theta(-[E_m - V_0]) \right] \\ \eta_{1,n} \left(B_n^o \left(\frac{1}{k_x n} \right) e^{ik_x n x} + B_n^i \left(-\frac{1}{k_x n} \right) e^{-ik_x n x} \right) \end{cases} \quad (54)$$

which looks the same as the non-tilted one, but here the momentum k_x and q_m are adjusted.

Through observation we can note that the qualitative effect tilting has on the x -momentum is to lower the energy contribution, similar to how the k_y^2 term interacts. Although, for only a certain range is the overall effect a decreased value under the square root. Strong enough tilting has the effect of increasing the energy term. For the tilt to have a similarly strong impact as the k_y^2 term implies the tilting must be comparable to $\tau_y \sim 2\mu k_y/\hbar$ which for the parameters we consider is $\tau_y \sim 10^4$. At the same time we have to consider proper values such that the incoming energy isn't corresponding to an evanescent mode rendering our calculations nonsensical. The evanescent region is described by

$$\begin{aligned} \frac{E_n}{\hbar k_y} - \frac{\hbar k_y}{2\mu} &< \tau_y < \frac{E_n}{\hbar k_y} + \frac{\hbar k_y}{2\mu}, \quad k_y > 0 \\ \frac{E_n}{\hbar k_y} - \frac{\hbar k_y}{2\mu} &> \tau_y > \frac{E_n}{\hbar k_y} + \frac{\hbar k_y}{2\mu}, \quad k_y < 0 \end{aligned}$$

which suggests the possibility of promoting previously evanescent modes to propagating modes, thus shifting the sidebands, by tuning the tilt to lie outside of the above bound.

For the S-matrix, as previously mentioned, we can adopt the same results as in Section VII due to no explicit change in the wavefunction. This means we consider the same matrix expressions as in Eq.(51) and end up with the same underlying expression for the S-matrix in Eq.(53).

Transmission is calculated with momentum constant at $k_y = 0.00095 \text{ \AA}^{-1}$, the smallest value in the Section VII, and varying τ_y on the order of $\sim 10^4$, shown in Fig.[8].

Comparing to the non-tilted case in Fig.[6] the Fano peaks are shifted similarly for the chosen parameters. Indeed, for the specific range of parameters chosen, varying

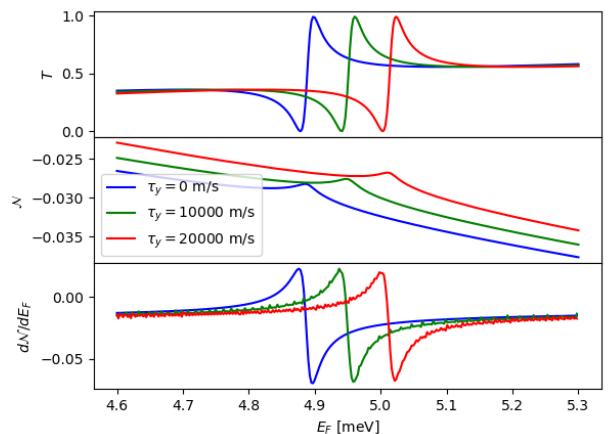


FIG. 8: Constant $k_y = 0.00095 \text{ \AA}^{-1}$. Fano peaks shift similar to no tilt and varying k_y . These peaks are captured in the shot-noise and more so in its differential.

momentum on the order of $\sim 10^5$ or tilt on the order of $\sim 10^4$ affects the scattering probabilities equally. This suggests some equivalence between momentum and tilt in their capacities as good quantum numbers.

IX. CONCLUSIONS

To learn how to work with non-adiabatic oscillating potential, in the framework of Floquet theory, the work of Refs.[2–4] were reproduced in regards to finding the Fano resonances present in transmission spectra. The pumped shot-noise, with use for measuring the resonances, has been elaborated upon for pedagogical reasons so that implementations in computational analysis is easier. A

detailed general derivation for diagonalizing a two-level Hamiltonian was made, and subsequently utilized.

The theory for a QBT-point system was extended to include linear tilt, enriching the structure. Two regions of tilt were identified that could promote evanescent modes to propagating and vice versa. Tilting on the order of $2\mu k_y/\hbar$ was shown to have similar effect on the spectrum as k_y . We therefore conclude that τ_y is also a good quantum number, with one magnitude less than that of k_y required for similar effects. This provides another control parameter for placing the resonant points as desired in applications.

For further research, considering negative tilt and momentum, but also sampling other parameter regions to apply the same analysis around the different types of

resonances as described in Ref.[4] is of interest to form a complete picture of the used setup. Generalizing tilt to incorporate both y and x tilt is a natural extension. This was pursued at first for this report, but proved to be too complicated to properly study given the time. Incorporating magnetic potentials, both static and oscillatory would complete the basic quantum pump part to enable circularly polarized light. One could also tilt Graphene and compare it with the tilted QBT system. This might provide more fundamental insights into the consequences of tilt, as it would compare a linear system (Graphene) to that of a quadratic (QBT). Natural results to compare, as extensions of this report, would be whether the shift in resonant energies are system dependent or not.

- [1] T. Oka and S. Kitamura, *Floquet Engineering of Quantum Materials*, Ann. Rev. Condens. Matt. Phys. **10**, 387408 (2019)
- [2] W. Li and L. E. Reichl, *Floquet scattering through a time-periodic potential*, Phys. Rev. B **60**, 15732-15741 (1999).
- [3] R. Zhu, J-H. Dai and Y. Guo, *Fano resonance in the nonadiabatically pumped shot noise of a time-dependent quantum well in a two-dimensional electron gas and graphene*, J. App. Phys. **117**, 164306 (2015).
- [4] S. Bera and I. Mandal, *Floquet scattering of quadratic band-touching semimetals through a time-periodic potential well*, J. Phys. Condens. Matter **33**, 295502 (2021).
- [5] J. H. Shirley, *Solution of the Schrödinger Equation with a Hamiltonian Periodic in Time*, Phys. Rev. **138**, B979-B987 (1965).
- [6] G. P. Berman, E. N. Bulgakov, D. K. Campbell, and A. F. Sadreev, *Resonant tunneling in time-periodically modulated semiconductor nanostructures*, Physica B: Condens. Matter **225**, 1 (1996).
- [7] E. N. Bulgakov and A. F. Sadreev, *Current - voltage characteristics of the resonant tunnelling double-barrier structure under time-periodical perturbation*, J. Phys.: Condens. Matter **8**, 8869 (1996).
- [8] M. Moskalets and M. Büttiker, *Floquet scattering theory for current and heat noise in large amplitude adiabatic pumps*, Phys. Rev. B **70**, 245305 (2004)

Appendix A: Numerics

	N	L [Å]	V_0 [meV]	V_1 [meV]	$\hbar\omega$ [meV]	μ [m_e]	v_F [m/s]	E_F [meV]	E_F steps	E range	E steps
1D	5	10	-20	5	1	0.067		[0, 1.4]	1000		
2DEG	5	10	-20	5	1	0.067		[0, 1.4]	1000	[0, E_F]	1000
Graphene	2	3000	-50	1			$c/300$	[7.2, 8.2]	200	[0, E_F]	30000
QBT	2	3000	-10	1	4	0.001		[4.6, 5.3]	200	[0, E_F]	500
QBT + tilt	2	3000	-10	1	4	0.001		[4.6, 5.3]	200	[3.5, E_F]	500

TABLE I: Numerical parameters

Appendix B: Code

1. 1D

```

1 import numpy as np
2 import scipy.special as spec
3 import scipy.constants as const
4 import matplotlib.pyplot as plt
5
6 def barrier_wave_vec(m, E_F, mu, V_0, hbaromega):
7     return np.sqrt((E_F+m*hbaromega-V_0)*2*mu*1.6*10**(-22))/(const.hbar)

```

```

8 def free_wave_vec(n, E_0, mu, hbaromega):
9     return np.sqrt((E_0+n*hbaromega)*2*mu*1.6*10**(-22)+0j)/(const.hbar)
10
11
12 def M_S(k, q, L, Bessel):
13     M_s_m = ((k[:,np.newaxis]+q[np.newaxis,:,:])*np.exp(-1j*q*L/2)-(k[:,np.newaxis]-q[np.newaxis,:,:])*np.exp(1j*q*L/2))*Bessel
14     M_s_p = ((k[:,np.newaxis]+q[np.newaxis,:,:])*np.exp(-1j*q*L/2)+(k[:,np.newaxis]-q[np.newaxis,:,:])*np.exp(1j*q*L/2))*Bessel
15     return M_s_m, M_s_p
16
17 def M_R(k, L):
18     M_r = np.diag(2*k*np.exp(-1j*k*L/2))
19     return M_r
20
21 def M_C(k, q, L, Bessel):
22     M_c_m = np.exp(-1j*(k[:,np.newaxis]-q[np.newaxis,:,:])*L/2)*Bessel
23     M_c_p = np.exp(-1j*(k[:,np.newaxis]+q[np.newaxis,:,:])*L/2)*Bessel
24     return M_c_m, M_c_p
25
26 def M_I(k, L):
27     return np.diag(np.exp(-1j*k*L/2))
28
29 def M_BA(M_c_m, M_c_p, M_r, M_s_m, M_s_p):
30     M_s_p_inv = np.linalg.inv(M_s_p)
31     M_s_m_inv = np.linalg.inv(M_s_m)
32     return 0.5*np.matmul(np.matmul(M_c_m,M_s_p_inv+M_s_m_inv)+np.matmul(M_c_p,M_s_p_inv-M_s_m_inv),M_r)
33
34 def T(M_BA, m, N):
35     return np.sum(np.absolute(M_BA[N:, m+N])**2)
36
37 def t(M_BA, n, m, N):
38     return np.absolute(M_BA[n+N, m+N])**2
39
40 def main():
41     mu = 0.067*const.m_e #Effective electron mass in GaAs in units of m_e
42     V_0 = -20 #meV
43     V_1 = 5 #meV
44     hbaromega = 1 #meV
45     L = 10*10**(-10) #m
46     N = 5 # Dictated by N>V_1/hbaromega
47     n = np.arange(-N,N+1) # Sideband indexing
48     Bessel = spec.jv(n[:,np.newaxis]-n[np.newaxis,:], V_1/hbaromega+0j) # The bessel function of
        the first kind, over all index combinations
49     E = np.linspace(0,1.4,1000) # Incoming energy in meV
50     t_00 = []
51     t_m10 = []
52     t_tot = []
53     for E_0 in E:
54         # Create the wave-vectors
55         k = free_wave_vec(n, E_0, mu, hbaromega)
56         q = barrier_wave_vec(n, E_0, mu, V_0, hbaromega)
57
58         # Create the first batch of matrices, _p and _m denotes + or - type (plus, minus)
59         M_c_m, M_c_p = M_C(k, q, L, Bessel)
60         M_r = M_R(k, L)
61         M_s_m, M_s_p = M_S(k, q, L, Bessel)
62
63         # Create the S-matrix submatrix M_BA we are interested in
64         matrix = M_BA(M_c_m, M_c_p, M_r, M_s_m, M_s_p)
65
66         t_00.append(t(matrix, 0, 0, N))
67         t_m10.append(t(matrix, -1, 0, N))
68         t_tot.append(T(matrix, 0, N))
69
70     fig, ax = plt.subplots(3,1)
71
72     # Plot transmissions
73     ax[0].plot(E, t_tot)

```

```

74 ax[0].set_ylabel('$T$')
75 ax[1].plot(E, t_00)
76 ax[1].set_ylabel('$|t_{0,0}|^2$')
77 ax[2].plot(E, t_m10)
78 ax[2].set_ylabel('$|t_{-1,0}|^2$')
79 ax[2].set_xlabel('E [meV]')
80 plt.show()
81 main()

```

2. 2DEG

```

1 import numpy as np
2 import scipy.special as spec
3 import scipy.constants as const
4 import matplotlib.pyplot as plt
5
6 def barrier_wave_vec(m, E_F, mu, V_0, hbaromega, ky):
7     return np.sqrt((E_F+m*hbaromega-V_0)*2*mu*1.6*10**(-22)/(const.hbar**2)+0j-(ky)**2)
8
9 def free_wave_vec(n, E_0, mu, hbaromega, ky):
10    return np.sqrt((E_0+n*hbaromega)*2*mu*1.6*10**(-22)/(const.hbar**2)+0j-(ky)**2)
11
12 def Matrices(k, q, L, Bessel):
13     M_c_m, M_c_p = M_C(k, q, L, Bessel)
14     M_r = M_R(k, L)
15     M_s_m, M_s_p = M_S(k, q, L, Bessel)
16     M_i = M_I(k, L)
17     return M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i
18
19 def M_S(k, q, L, Bessel):
20     M_s_m = ((k[:, np.newaxis]+q[np.newaxis,:,:])*np.exp(-1j*q*L/2)-(k[:, np.newaxis]-q[np.newaxis,:,:])*np.exp(1j*q*L/2))*Bessel
21     M_s_p = ((k[:, np.newaxis]+q[np.newaxis,:,:])*np.exp(-1j*q*L/2)+(k[:, np.newaxis]-q[np.newaxis,:,:])*np.exp(1j*q*L/2))*Bessel
22     return M_s_m, M_s_p
23
24 def M_R(k, L):
25     return np.diag(2*k*np.exp(-1j*k*L/2))
26
27 def M_C(k, q, L, Bessel):
28     M_c_m = np.exp(-1j*(k[:, np.newaxis]-q[np.newaxis,:,:])*L/2)*Bessel
29     M_c_p = np.exp(-1j*(k[:, np.newaxis]+q[np.newaxis,:,:])*L/2)*Bessel
30     return M_c_m, M_c_p
31
32 def M_I(k, L):
33     return np.diag(np.exp(-1j*k*L/2))
34
35 def Matrix(M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i, k):
36
37     # Produce the inverses
38     M_s_p_inv = np.linalg.inv(M_s_p)
39     M_s_m_inv = np.linalg.inv(M_s_m)
40
41     # Compute the normalizing factor (Re(kn)/Re(km))^-1/2
42     x1 = np.real(k[:, np.newaxis])
43     x2 = np.real(k[np.newaxis, :])
44     norm = np.where(x2 == 0, 0, np.sqrt(x1/x2))
45
46     # Combine the inverses accodingly
47     pM_s = M_s_p_inv+M_s_m_inv
48     mM_s = M_s_p_inv-M_s_m_inv
49
50     # Compute the final reflection and transmission matrices
51     m_AA = 0.5*norm*np.matmul(np.matmul(M_c_p, pM_s)+np.matmul(M_c_m, mM_s), M_r)-norm*M_i #Reflection
52         from either side
      m_AB = 0.5*norm*np.matmul(np.matmul(M_c_p, mM_s)+np.matmul(M_c_m, pM_s), M_r) #Transmission to
          left

```

```

53     m_BA = 0.5*norm*np.matmul(np.matmul(M_c_m,pM_s)+np.matmul(M_c_p,mM_s),M_r) #Transmission to
54     right
55     return m_AA, m_AB, m_BA, m_AA
56
56 def altf(n, hw, E, E_F): # make into a matrix with 0 and 1 to multiply the entire matrix M_nmp
57     return (0.5*np.sign(n*hw + E-E_F)[:,np.newaxis]-0.5*np.sign(n*hw + E-E_F)[np.newaxis, :])**2
58
59 def shot_noise(N, step, sE_RR, sE_RL, s_LR, s_LL):
60     S_r1 = 0
61     ran = range(-N,N+1)
62     for n in ran:
63         for m in ran:
64             for p in ran:
65                 for e in range(2*N+1):
66                     M_RLRR = np.conj(sE_RR[e,n+N])*sE_RR[e,m+N]*np.conj(s_LR[p,m+N])*s_LR[p,n+N]
67                     M_RRLR = np.conj(sE_RR[e,n+N])*sE_RL[e,m+N]*np.conj(s_LL[p,m+N])*s_LL[p,n+N]
68                     M_RLLR = np.conj(sE_RL[e,n+N])*sE_RR[e,m+N]*np.conj(s_LR[p,m+N])*s_LL[p,n+N]
69                     M_RLLL = np.conj(sE_RL[e,n+N])*sE_RL[e,m+N]*np.conj(s_LL[p,m+N])*s_LL[p,n+N]
70
71                     M_mnp = M_RLRR + M_RRLR + M_RLLR + M_RLLL
72
73                     S_r1 += M_mnp*step*fermi(n,m)
74     return S_r1
75
76 def Transmission(s_RL, m, N):
77     return np.sum(np.absolute(s_RL[N:, m+N]))**2
78
79 def ddE(A,E_F):
80     return np.diff(A)/np.diff(E_F)
81
82 def main():
83     mu = 0.067*const.m_e #Effective electron mass in GaAs in units of m_e
84     V_0 = -20 #meV
85     V_1 = 5 #meV
86     hbaromega = 1 #meV
87     L = 10*10**(-10) # m
88     N = 5 # Dictated by N>V_1/hbaromega
89     n = np.arange(-N,N+1) # Sideband indexing
90     Bessel = spec.jv(n[:,np.newaxis]-n[np.newaxis,:], V_1/hbaromega) # The bessel function of the
91     first kind, over all index combinations
92     E_F = np.linspace(0,1.4,1000) # Incoming energy in meV
93     dE = (N+1)/(2*N+1)
94     E = np.arange(0, N+1, dE)
95     fig, ax = plt.subplots(1,1)
96     kys = [0, 0.001, 0.002]
97     cols = ['black', 'red', 'blue']
98     for i, ky in enumerate(kys):
99         ky /= 10**(-10)
100        T = []
101        S_RL = []
102        S_LL = []
103        for E_0 in E_F:
104            k = free_wave_vec(n, E_0, mu, hw, ky)
105            q = barrier_wave_vec(n, E_0, mu, V_0, hw, ky)
106            M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i = Matrices(k, q, L, Bessel)
107            s_LL, s_LR, s_RL, s_RR = Matrix(M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i, k)
108            T.append(Transmission(s_LR, 0, N))
109
110        S_ll = 0
111        Emax = E_0
112        Esteps = 1000
113        E = np.linspace(0, Emax, Esteps)
114        dE = Emax/Esteps
115        for e in E:
116            F = altf(n[N:], hw, e, E_0)
117            kE = free_wave_vec(n, e, mu, hw, ky)
118            qE = barrier_wave_vec(n, e, mu, V_0, hw, ky)
119
120            ME_c_m, ME_c_p, ME_r, ME_s_m, ME_s_p, ME_i = Matrices(kE, qE, L, Bessel)
```

```

121     sE_LL, sE_LR, sE_RL, sE_RR = Matrix(ME_c_m, ME_c_p, ME_r, ME_s_m, ME_s_p, ME_i, kE)
122
123     S_LL += Shot_noise(N, F, e, E_0, sE_LR, sE_LL)
124     S_LL.append(S_LL*dE)
125     dS_LL = ddE(S_LL, E_F)
126
127     ax[0].plot(E_F, T, c = cols[i], label=f'k${}_y${}={kys[i]} 1/ ')
128     ax[1].plot(E_F, S_LL, c = cols[i], label=f'k${}_y${}={kys[i]} 1/ ')
129     ax[2].plot(E_F[:-1], dS_LL, c = cols[i], label=f'k${}_y${}={kys[i]} 1/ ')
130
131     ax[0].set_ylabel('$T$')
132     ax[1].set_ylabel('$\mathcal{N}$')
133     ax[2].set_ylabel('$d\mathcal{N}/dE_F$')
134     ax[1].legend()
135     ax[2].set_xlabel('$E_F$ [meV]')
136     plt.show()
137 main()

```

3. Graphene

```

1 import numpy as np
2 import scipy.special as spec
3 import scipy.constants as const
4 import matplotlib.pyplot as plt
5
6 def barrier(m, E_F, hvf, V_0, hbaromega, ky):
7     q = np.sqrt(((E_F+m*hbaromega-V_0)/hvf)**2+0j-ky**2)
8     theta = np.where(np.real(q)==0, np.arctan(ky/q)-np.pi, np.arctan(ky/q))
9     sm = np.sign(E_F+m*hbaromega-V_0)
10    return q, theta, sm
11
12 def free(n, E_0, hvf, hbaromega, ky):
13     k = np.sqrt(((E_0+n*hbaromega)/hvf)**2+0j-ky**2)
14     phi = np.where(np.real(k)==0, np.arctan(ky/k)-np.pi, np.arctan(ky/k))
15     sn = np.sign(E_0+n*hbaromega)
16     return k, phi, sn
17
18 def Matrices(k, q, L, Bessel, sn, sm, phi, theta):
19     M_c_m, M_c_p = M_C(k, q, L, Bessel)
20     M_r = M_R(k, L, phi, sn)
21     M_sa_m, M_sa_p = M_Sa(q, L, Bessel, sn, sm, phi, theta)
22     M_sb_m, M_sb_p = M_Sb(q, L, Bessel, sn, sm, phi, theta)
23     M_i = M_I(k, L)
24     return M_c_m, M_c_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i
25
26 def M_Sa(q, L, Bessel, sn, sm, phi, theta):
27     M_sa_m = ((sm[np.newaxis,:]*np.exp(1j*theta[np.newaxis,:]))+sn[:,np.newaxis]*np.exp(-1j*phi[:,np.newaxis]))*np.exp(-1j*q[np.newaxis,:]*L/2)-(sn[:,np.newaxis]*np.exp(1j*phi[:,np.newaxis])-sm[np.newaxis,:]*np.exp(1j*theta[np.newaxis,:]))*np.exp(1j*q[np.newaxis,:]*L/2))*Bessel
28     M_sa_p = ((sm[np.newaxis,:]*np.exp(1j*theta[np.newaxis,:]))+sn[:,np.newaxis]*np.exp(-1j*phi[:,np.newaxis]))*np.exp(-1j*q[np.newaxis,:]*L/2)+(sn[:,np.newaxis]*np.exp(1j*phi[:,np.newaxis])-sm[np.newaxis,:]*np.exp(1j*theta[np.newaxis,:]))*np.exp(1j*q[np.newaxis,:]*L/2))*Bessel
29     return M_sa_m, M_sa_p
30
31 def M_Sb(q, L, Bessel, sn, sm, phi, theta):
32     M_sb_m = ((sn[:,np.newaxis]*np.exp(-1j*phi[:,np.newaxis])-sm[np.newaxis,:]*np.exp(-1j*theta[np.newaxis,:]))*np.exp(1j*q[np.newaxis,:]*L/2)-(sn[:,np.newaxis]*np.exp(1j*phi[:,np.newaxis])+sm[np.newaxis,:]*np.exp(-1j*theta[np.newaxis,:]))*np.exp(-1j*q[np.newaxis,:]*L/2))*Bessel
33     M_sb_p = ((sn[:,np.newaxis]*np.exp(-1j*phi[:,np.newaxis])-sm[np.newaxis,:]*np.exp(-1j*theta[np.newaxis,:]))*np.exp(1j*q[np.newaxis,:]*L/2)+(sn[:,np.newaxis]*np.exp(1j*phi[:,np.newaxis])+sm[np.newaxis,:]*np.exp(-1j*theta[np.newaxis,:]))*np.exp(-1j*q[np.newaxis,:]*L/2))*Bessel
34     return M_sb_m, M_sb_p
35
36 def M_R(k, L, phi, sn):
37     return np.diag(2*np.cos(phi)*sn*np.exp(-1j*k*L/2))
38
39 def M_C(k, q, L, Bessel):

```

```

40 M_c_m = np.exp(-1j*(k[:,np.newaxis]-q[np.newaxis,:,])*L/2)*Bessel
41 M_c_p = np.exp(-1j*(k[:,np.newaxis]+q[np.newaxis,:])*L/2)*Bessel
42 return M_c_m, M_c_p
43
44 def M_I(k, L):
45     return np.diag(np.exp(-1j*k*L/2))
46
47 def Matrix(M_c_m, M_c_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i, k):
48
49     # Produce the inverses
50     M_sa_p_inv = np.linalg.inv(M_sa_p)
51     M_sa_m_inv = np.linalg.inv(M_sa_m)
52     M_sb_p_inv = np.linalg.inv(M_sb_p)
53     M_sb_m_inv = np.linalg.inv(M_sb_m)
54
55     # Compute the normalizing factor (Re(kn)/Re(km))^-1/2
56     x1 = np.real(k[:,np.newaxis])
57     x2 = np.real(k[np.newaxis,:])
58     norm = np.where(x2 == 0, 0, np.sqrt(x1/x2))
59
60     # Combine the inverses accordingly
61     pM_sa = M_sa_p_inv+M_sa_m_inv
62     mM_sa = M_sa_p_inv-M_sa_m_inv
63     pM_sb = M_sb_p_inv+M_sb_m_inv
64     mM_sb = M_sb_p_inv-M_sb_m_inv
65
66     # Combine other stuff
67     appmm = np.matmul(M_sb_p_inv,M_sa_p)-np.matmul(M_sb_m_inv, M_sa_m)
68     bppmm = np.matmul(M_sa_p_inv,M_sb_p)-np.matmul(M_sa_m_inv, M_sb_m)
69
70     # Get the inverses
71     appmm_inv = np.linalg.inv(appmm)
72     bppmm_inv = np.linalg.inv(bppmm)
73
74     # Compute parts of final
75     aA = np.matmul(np.matmul(appmm_inv, mM_sb), M_r)
76     aB = np.matmul(np.matmul(appmm_inv, pM_sb), M_r)
77     bA = np.matmul(np.matmul(bppmm_inv, mM_sa), M_r)
78     bB = np.matmul(np.matmul(bppmm_inv, pM_sa), M_r)
79
80     # Compute the final reflection and transmission matrices
81     m_AA = norm*(np.matmul(M_c_p, aA)+np.matmul(M_c_m, bA)-M_i) #Reflection from left
82     m_AB = norm*(np.matmul(M_c_p, aB)+np.matmul(M_c_m, bB)) #Transmission to left
83     m_BA = norm*(np.matmul(M_c_m, aA)+np.matmul(M_c_p, bA)) #Transmission to right
84     m_BB = norm*(np.matmul(M_c_m, aB)+np.matmul(M_c_p, bB)-M_i) #Reflection from right
85     return m_AA, m_AB, m_BA, m_BB
86
87
88 def altfermi(n, hw, E, E_F): # make into a matrix with 0 and 1 to multiply the entire matrix M_nmp
89     return 0.5*(np.sign(E-E_F+n[:, np.newaxis]*hw)-np.sign(E-E_F+n[np.newaxis, :]*hw))**2
90
91 def herm(A):
92     return np.transpose(np.conjugate(A))
93
94 def shot_noise(N, F, sE_LR, sE_LL):
95     sELR = sE_LR[N:, N:]
96     sELL = sE_LL[N:, N:]
97
98     M_LLRR = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis,:]), (F*np.
99         matmul(herm(sELR),sELR)))
100    M_LLRL = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis,:]), (F*np.
101        matmul(herm(sELL),sELR)))
102    M_LLLR = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis,:]), (F*np.
103        matmul(herm(sELR),sELL)))
104    M_LLLL = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis,:]), (F*np.
105        matmul(herm(sELL),sELL)))
106
107    M_nmp = M_LLRR+M_LLRL+M_LLLR+M_LLLL
108    return np.trace(M_nmp)
109
110

```

```

106 def Transmission(s_RL, m, N):
107     return np.sum(np.absolute(s_RL[m+N:, N:])**2)
108
109 def dDE(A, E_F):
110     return np.diff(A)/np.diff(E_F)
111
112 def main():
113     V_0 = -50 #meV
114     V_1 = 1 #meV
115     mev = 1.602*(10**(-19))/(10**3)
116     hvf = (10**6)*const.hbar/mev #10**6*const.hbar*6.2447*10**18
117     hw = 4 #meV
118     L = 3000*(10**(-10)) # m
119     N = 2 # Dictated by N>V_1/hbaromega
120     n = np.arange(-N,N+1) # Sideband indexing
121     m = np.arange(N+1)
122     Bessel = spec.jv(n[:,np.newaxis]-n[np.newaxis,:], V_1/hw) # The bessel function of the first
123     kind, over all index combinations
124     E_F = np.linspace(7.2, 8.2, 200) # Incoming energy in meV
125     Esteps = [30000, 10000, 3000]
126     kys = [0.0008, 0.0009, 0.001]
127     cols = ['black', 'red', 'blue']
128     fig, ax = plt.subplots(3,1,sharex=True)
129     fig.subplots_adjust(hspace=0)
130     for i, ky in enumerate(kys):
131         ky *= 10**(10)
132         T = []
133         S_LL = []
134         for E_0 in E_F:
135             k, phi, sn = free(n, E_0, hvf, hw, ky)
136             q, theta, sm = barrier(n, E_0, hvf, V_0, hw, ky)
137
138             M_c_m, M_c_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i = Matrices(k, q, L, Bessel, sn,
139             sm, phi, theta)
140
141             s_LL, s_LR, s_RL, s_RR = Matrix(M_c_m, M_c_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i,
142             k)
143
144             T.append(Transmission(s_RL, 0, N))
145
146             S_ll = 0
147             Emax = E_0 #+ N*hw
148             E = np.linspace(0.001, Emax, Esteps[i])
149             dE = Emax/Esteps[i]
150
151             for e in E:
152                 F = altfermi(m, hw, e, E_0)
153                 kE, phiE, snE = free(n, e, hvf, hw, ky)
154                 qE, thetaE, smE = barrier(n, e, hvf, V_0, hw, ky)
155
156                 ME_c_m, ME_c_p, ME_r, ME_sa_m, ME_sa_p, ME_sb_m, ME_sb_p, ME_i = Matrices(kE, qE, L,
157                 Bessel, snE, smE, phiE, thetaE)
158
159                 sE_LL, sE_LR, sE_RL, sE_RR = Matrix(ME_c_m, ME_c_p, ME_r, ME_sa_m, ME_sa_p, ME_sb_m,
160                 ME_sb_p, ME_i, kE)
161
162                 S_ll += shot_noise(N, F, sE_LR, sE_LL)
163
164                 S_LL.append(S_ll*dE)
165                 dS_LL = ddE(S_LL, E_F)
166
167                 ax[0].plot(E_F, T, c = cols[i])
168                 ax[1].plot(E_F, S_LL, c = cols[i], label=f'k${}_y$={kys[i]} 1/ ')
169                 ax[2].plot(E_F[:-1], dS_LL, c = cols[i])
170
171                 ax[0].set_ylabel('$T$')
172                 ax[1].set_ylabel('$\mathcal{N}$')
173                 ax[1].legend()
174                 ax[2].set_ylabel('$d\mathcal{N}/dE_F$')
175                 ax[2].set_xlabel('$E_F$ [meV]')

```

```
171 plt.show()
```

4. QBT

```

1 import numpy as np
2 import scipy.special as spec
3 import scipy.constants as const
4 import matplotlib.pyplot as plt
5
6 def barrier(m, E_F, mu, V_0, hw, ky):
7     q = np.sqrt(2*mu*(1.602*10**(-22)))*np.abs(E_F+m*hw-V_0)/const.hbar**2+0j-ky**2)
8     Theta2 = np.heaviside(E_F+m*hw, 0.5)
9     Theta3 = np.heaviside(-E_F-m*hw, 0.5)
10    gamma2 = q/np.sqrt(q**2+ky**2)
11    gamma3 = ky/np.sqrt(q**2+ky**2)
12    return q, Theta2, Theta3, gamma2, gamma3
13
14 def free(n, E_0, mu, hw, ky):
15     k = np.sqrt(2*mu*(1.602*10**(-22)))*(E_0+n*hw)/const.hbar**2+0j-ky**2)
16     gamma1 = k/np.sqrt(k**2+ky**2)
17     return k, gamma1
18
19 def Matrices(k, q, ky, L, gamma1, gamma2, gamma3, E_F, m, hw, Bessel, Theta2, Theta3):
20     M_c_m, M_c_p = M_C(k, q, L, gamma1, gamma2, gamma3, E_F, m, hw, Bessel, Theta2, Theta3)
21     M_r = M_R(k, L, gamma1)
22     M_s_m, M_s_p = M_S(k, q, ky, L, gamma1, gamma2, gamma3, E_F, m, hw, Bessel, Theta2, Theta3)
23     M_i = M_I(k, L)
24     return M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i
25
26 def M_S(k, q, ky, L, gamma1, gamma2, gamma3, E_F, m, hw, Bessel, Theta2, Theta3):
27     M_s_m = (Theta2[np.newaxis,:]*gamma2[np.newaxis,:]*(1+k[:,np.newaxis]/q[np.newaxis,:])*np.exp
28             (-1j*q[np.newaxis,:]*L/2)-(1-k[:,np.newaxis]/q[np.newaxis,:])*np.exp(1j*q[np.newaxis,:]*L/2))+Theta3[np.newaxis,:]*gamma3[np.newaxis,:]*(1-k[:,np.newaxis]*q[np.newaxis,:]/ky**2)*np.exp(-1j*q[np.newaxis,:]*L/2)-(1+k[:,np.newaxis]*q[np.newaxis,:]/ky**2)*np.exp(1j*q[np.newaxis,:]*L/2))*Bessel
29     M_s_p = (Theta2[np.newaxis,:]*gamma2[np.newaxis,:]*(1+k[:,np.newaxis]/q[np.newaxis,:])*np.exp
30             (-1j*q[np.newaxis,:]*L/2)+(1-k[:,np.newaxis]/q[np.newaxis,:])*np.exp(1j*q[np.newaxis,:]*L/2))+Theta3[np.newaxis,:]*gamma3[np.newaxis,:]*(1-k[:,np.newaxis]*q[np.newaxis,:]/ky**2)*np.exp(-1j*q[np.newaxis,:]*L/2)+(1+k[:,np.newaxis]*q[np.newaxis,:]/ky**2)*np.exp(1j*q[np.newaxis,:]*L/2))*Bessel
31     return M_s_m, M_s_p
32
33 def M_R(k, L, gamma1):
34     return np.diag(2*gamma1*np.exp(-1j*k*L/2))
35
36 def M_C(k, q, L, gamma1, gamma2, gamma3, E_F, m, hw, Bessel, Theta2, Theta3):
37     M_c_m = (gamma2[np.newaxis,:]*Theta2[np.newaxis,:]+gamma2[np.newaxis,:]*Theta3[np.newaxis,:])*np.exp(-1j*(k[:,np.newaxis]-q[np.newaxis,:])*L/2)*Bessel/gamma1[:,np.newaxis]
38     M_c_p = (gamma2[np.newaxis,:]*Theta2[np.newaxis,:]+gamma2[np.newaxis,:]*Theta3[np.newaxis,:])*np.exp(-1j*(k[:,np.newaxis]+q[np.newaxis,:])*L/2)*Bessel/gamma1[:,np.newaxis]
39     return M_c_m, M_c_p
40
41 def M_I(k, L):
42     return np.diag(np.exp(-1j*k*L))
43
44 def Matrix(M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i, k):
45     # Produce the inverses
46     M_s_p_inv = np.linalg.inv(M_s_p)
47     M_s_m_inv = np.linalg.inv(M_s_m)
48
49     # Compute the normalizing factor (Re(kn)/Re(km))^-1/2
50     x1 = np.real(k[:,np.newaxis])
51     x2 = np.real(k[np.newaxis,:])
52
53     norm = np.where(x2 == 0, 0, np.sqrt(x1/x2))
#norm = np.sqrt(x1/x2)
```

```

54 #norm = 1
55
56 # Combine the inverses accordingly
57 pM_s = M_s_p_inv+M_s_m_inv
58 mM_s = M_s_p_inv-M_s_m_inv
59
60 # Combine other stuff
61 aA = 0.5*np.matmul(pM_s, M_r)
62 aB = 0.5*np.matmul(mM_s, M_r)
63 #bA = aB
64 #bB = aA
65
66
67 # Compute the final reflection and transmission matrices
68 m_AA = norm*(np.matmul(M_c_p, aA)+np.matmul(M_c_m, aB)-M_i) #Reflection from left
69 m_AB = norm*(np.matmul(M_c_p, aB)+np.matmul(M_c_m, aA)) #Transmission to left
70 m_BA = norm*(np.matmul(M_c_m, aA)+np.matmul(M_c_p, aB)) #Transmission to right
71 m_BB = norm*(np.matmul(M_c_m, aB)+np.matmul(M_c_p, aA)-M_i) #Reflection from right
72 return m_AA, m_AB, m_BA, m_BB
73
74 def altfermi(n, hw, E, E_F): # make into a matrix with 0 and 1 to multiply the entire matrix M_nmp
75     return 0.5*(np.sign(E-E_F+n[:, np.newaxis]*hw)-np.sign(E-E_F+n[np.newaxis, :]*hw))**2
76
77 def herm(A):
78     return np.transpose(np.conjugate(A))
79
80 def shot_noise(N, F, sE_LR, sE_LL):
81     sELR = sE_LR[N:, N:]
82     sELL = sE_LL[N:, N:]
83
84     M_LLRR = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis, :]), (F*np.
85         matmul(herm(sELR), sELR)))
86     M_LLRL = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis, :]), (F*np.
87         matmul(herm(sELL), sELR)))
88     M_LLLR = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis, :]), (F*np.
89         matmul(herm(sELR), sELL)))
90     M_LLLL = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis, :]), (F*np.
91         matmul(herm(sELL), sELL)))
92
92     M_nmp = M_LLRR+M_LLRL+M_LLLR+M_LLLL
93     return np.trace(M_nmp)
94
95 def Transmission(s_RL, m, N):
96     return np.sum(np.absolute(s_RL[m+N, N:])**2)
97
98 def ddE(A, E_F):
99     return np.diff(A)/np.diff(E_F)
100
101 def main():
102     V_0 = -10 #meV
103     V_1 = 1 #meV
104     #mev = 1.602*(10**(-19))/(10**3)
105     #mev = 1.602*10**(-22)
106     mu = 0.001*const.m_e
107     #hvf = (10**6)*const.hbar/mev #10**6*const.hbar*6.2447*10**18
108     hw = 4 #meV
109     L = 3000*(10**(-10)) # m
110     N = 2 # Dictated by N>V_1/hbaromega
111     n = np.arange(-N, N+1) # Sideband indexing
112     m = np.arange(N+1)
113     Bessel = spec.jv(n[:, np.newaxis]-n[np.newaxis, :], V_1/hw) # The bessel function of the first
114     # kind, over all index combinations
115     E_F = np.linspace(4.6, 5.3, 200) # Incoming energy in meV
116     fig, ax = plt.subplots(3,1,sharex=True)
117     fig.subplots_adjust(hspace=0)
118     kys = [0.00095, 0.00096, 0.00097]
119     cols = ['blue', 'green', 'red']
120     for i, ky in enumerate(kys):
121         ky *= 10**(10)
122         T = []

```

```

119     S_LL = []
120     for E_0 in E_F:
121         k, gamma1 = free(n, E_0, mu, hw, ky)
122         q, Theta2, Theta3, gamma2, gamma3 = barrier(n, E_0, mu, V_0, hw, ky)
123
124         M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i = Matrices(k, q, ky, L, gamma1, gamma2, gamma3,
125         E_F, m, hw, Bessel, Theta2, Theta3)
126
127         s_LL, s_LR, s_RL, s_RR = Matrix(M_c_m, M_c_p, M_r, M_s_m, M_s_p, M_i, k)
128
129         T.append(Transmission(s_RL, 0, N))
130
131         S_ll = 0
132         Emax = E_0
133         Esteps = 500
134         E = np.linspace(0.001, Emax, Esteps)
135         dE = Emax/Esteps
136         for e in E:
137             F = altfermi(m, hw, e, E_0)
138
139             kE, gamma1E = free(n, e, mu, hw, ky)
140             qE, Theta2E, Theta3E, gamma2E, gamma3E = barrier(n, e, mu, V_0, hw, ky)
141
142             ME_c_m, ME_c_p, ME_r, ME_s_m, ME_s_p, ME_i = Matrices(kE, qE, ky, L, gamma1E,
143             gamma2E, gamma3E, e, m, hw, Bessel, Theta2E, Theta3E)
144
145             sE_LL, sE_LR, sE_RL, sE_RR = Matrix(ME_c_m, ME_c_p, ME_r, ME_s_m, ME_s_p, ME_i, kE)
146
147             S_ll += shot_noise(N, F, sE_LR, sE_LL)
148
149             S_LL.append(S_ll*dE)
150             dS_LL = ddE(S_LL, E_F)
151
152             ax[0].plot(E_F, T, c = cols[i])
153             ax[1].plot(E_F, S_LL, c = cols[i], label=f'k${}_y$={kys[i]} 1/ ')
154             ax[2].plot(E_F[:-1], dS_LL, c = cols[i])
155
156             ax[0].set_ylabel('T')
157             ax[1].set_ylabel('$\mathcal{N}$')
158             ax[1].legend()
159             ax[2].set_ylabel('$d\mathcal{N}/dE_F$')
160             ax[2].set_xlabel('$E_F$ [meV]')
161             plt.show()

```

5. QBT with tilt

```

1 import numpy as np
2 import scipy.special as spec
3 import scipy.constants as const
4 import matplotlib.pyplot as plt
5
6 def barrier(m, E_F, mu, V_0, hw, ky, tau, tauy, mev):
7     E_m = mev*(E_F+m*hw-V_0)-const.hbar*tauy*ky # Joules
8     q1 = -tau+np.sqrt(tau**2+2*mu*np.abs(E_m)/const.hbar**2+0j-ky**2)
9     q2 = tau+np.sqrt(tau**2+2*mu*np.abs(E_m)/const.hbar**2+0j-ky**2)
10    Theta_p = np.heaviside(E_m-np.real(tau*const.hbar*q1/mu), 0)
11    Theta_m = np.heaviside(-E_m+np.real(tau*const.hbar*q1/mu), 0)
12    qTheta_p = np.heaviside(np.real(q1), 0)
13    qTheta_m = np.heaviside(-np.real(q1), 0)
14    return q1, q2, Theta_p, Theta_m, qTheta_p, qTheta_m
15
16 def free(n, E_0, mu, hw, ky, tau, tauy, mev):
17     E_n = mev*(E_0+n*hw)-const.hbar*tauy*ky
18     k1 = -tau+np.sqrt(tau**2+2*mu*np.abs(E_n)/const.hbar**2+0j-ky**2)
19     k2 = tau+np.sqrt(tau**2+2*mu*np.abs(E_n)/const.hbar**2+0j-ky**2)
20     #kTheta_p = np.heaviside(E_n-np.real(tau*const.hbar*k1/mu), 0)
21     return k1, k2#, kTheta_p

```

```

22
23 def varsigma1(ky, 1):
24     return ky/np.sqrt(np.abs(1)**2+ky**2)
25
26 def varsigma2(ky, 1):
27     return np.abs(1)/np.sqrt(np.abs(1)**2+ky**2)
28
29 def Matrices(k1, k2, q1, q2, ky, L, m, hw, Bessel, Theta_p, Theta_m, qTheta_p, qTheta_m,
30             sigma1k1, sigma1k2, sigma1q1, sigma1q2, sigma2q1, sigma2q2):
31     M_c_A_m, M_c_A_p, M_c_B_m, M_c_B_p = M_C(k1, k2, q1, q2, L, Bessel, Theta_p, Theta_m, qTheta_p,
32             qTheta_m,
33             sigma1k1, sigma1k2, sigma1q1, sigma1q2, sigma2q1,
34             sigma2q2)
35     M_r = M_R(k1, k2, L, sigma1k1, sigma1k2)
36     M_sa_m, M_sa_p, M_sb_m, M_sb_p = M_S(k1, k2, q1, q2, ky, L, m, hw, Bessel, Theta_p, Theta_m,
37             qTheta_p, qTheta_m,
38             sigma1k1, sigma1k2, sigma1q1, sigma1q2, sigma2q1, sigma2q2)
39
40 def M_S(k1, k2, q1, q2, ky, L, m, hw, Bessel, Theta_p, Theta_m, qTheta_p,
41         qTheta_m, sigma1k1, sigma1k2, sigma1q1, sigma1q2, sigma2q1, sigma2q2):
42
43     M_sa_p1 = ((1+q1[np.newaxis,:]/k1[:,np.newaxis])*np.exp(-1j*q1[np.newaxis,:]*L/2)
44                 +(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-q1[np.newaxis,:]/k2[:,np.newaxis])
45                 )*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q1[np.newaxis,:])*L/2)) #Klar
46
47     M_sa_p2 = ((1-ky**2/(k1[:,np.newaxis]*q2[np.newaxis,:]))*np.exp(-1j*q2[np.newaxis,:]*L/2)
48                 +(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1+ky**2/(k2[:,np.newaxis]*q2[np.newaxis,:])))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q2[np.newaxis,:])*L/2))
49
50     M_sa_p3 = ((1+ky**2/(k1[:,np.newaxis]*q1[np.newaxis,:]))*np.exp(1j*q1[np.newaxis,:]*L/2)
51                 +(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-ky**2/(k2[:,np.newaxis]*q1[np.newaxis,:])))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q1[np.newaxis,:])*L/2))
52
53     M_sa_p = Bessel*(Theta_p[np.newaxis,:]*qTheta_p[np.newaxis,:]*sigma1q1[np.newaxis,:]*M_sa_p1
54                 +Theta_m[np.newaxis,:]*sigma2q2[np.newaxis,:]*M_sa_p2
55                 +Theta_m[np.newaxis,:]*qTheta_m[np.newaxis,:]*sigma2q1[np.newaxis,:]*M_sa_p3)
# Klar
56
57     M_sa_m1 = ((1+q1[np.newaxis,:]/k1[:,np.newaxis])*np.exp(-1j*q1[np.newaxis,:]*L/2)
58                 -(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-q1[np.newaxis,:]/k2[:,np.newaxis])
59                 )*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q1[np.newaxis,:])*L/2))
60
61     M_sa_m2 = ((1-ky**2/(k1[:,np.newaxis]*q2[np.newaxis,:]))*np.exp(-1j*q2[np.newaxis,:]*L/2)
62                 -(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1+ky**2/(k2[:,np.newaxis]*q2[np.newaxis,:])))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q2[np.newaxis,:])*L/2))
63
64     M_sa_m3 = ((1+ky**2/(k1[:,np.newaxis]*q1[np.newaxis,:]))*np.exp(1j*q1[np.newaxis,:]*L/2)
65                 -(sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-ky**2/(k2[:,np.newaxis]*q1[np.newaxis,:])))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q1[np.newaxis,:])*L/2))
66
67     M_sa_m = Bessel*(Theta_p[np.newaxis,:]*qTheta_p[np.newaxis,:]*sigma1q1[np.newaxis,:]*M_sa_m1
68                 +Theta_m[np.newaxis,:]*sigma2q2[np.newaxis,:]*M_sa_m2
69                 +Theta_m[np.newaxis,:]*qTheta_m[np.newaxis,:]*sigma2q1[np.newaxis,:]*M_sa_m3)
# Klar
70
71     M_sb_p1 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1+q2[np.newaxis,:]/k2[:,np.newaxis])
72                 )*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q2[np.newaxis,:])*L/2)
73                 +(1-q2[np.newaxis,:]/k1[:,np.newaxis])*np.exp(1j*q2[np.newaxis,:]*L/2))
74
75     M_sb_p2 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-q1[np.newaxis,:]/k2[:,np.newaxis])
76                 )*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q1[np.newaxis,:])*L/2)
77                 +(1+q1[np.newaxis,:]/k1[:,np.newaxis])*np.exp(-1j*q1[np.newaxis,:]*L/2))
78
79     M_sb_p3 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-ky**2/(k2[:,np.newaxis]*q1[np.newaxis,:])))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q1[np.newaxis,:])*L/2)

```

```

77     +(1+ky**2/(k1[:,np.newaxis]*q1[np.newaxis,:,:]))*np.exp(1j*(q1[np.newaxis,:,:])*L/2))
78
79 M_sb_p = Bessel*(Theta_p[np.newaxis,:,:]*sigma1q2[np.newaxis,:,:]*M_sb_p1
80           +Theta_p[np.newaxis,:,:]*qTheta_m[np.newaxis,:,:]*sigma1q1[np.newaxis,:,:]*M_sb_p2
81           +Theta_m[np.newaxis,:,:]*qTheta_p[np.newaxis,:,:]*sigma2q1[:,np.newaxis]*M_sb_p3)
82
83
84 M_sb_m1 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1+q2[np.newaxis,:,:]/k2[:,np.newaxis])
85   ])*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q2[np.newaxis,:,:])*L/2)
86   -(1-q2[np.newaxis,:,:]/k1[:,np.newaxis])*np.exp(1j*q2[np.newaxis,:,:]*L/2))
87
88 M_sb_m2 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-q1[np.newaxis,:,:]/k2[:,np.newaxis]
89   ])*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]-q1[np.newaxis,:,:])*L/2)
90   -(1+q1[np.newaxis,:,:]/k1[:,np.newaxis])*np.exp(-1j*q1[np.newaxis,:,:]*L/2))
91
92 M_sb_m3 = ((sigma1k1[:,np.newaxis]/sigma1k2[:,np.newaxis])*(1-ky**2/(k2[:,np.newaxis])*q1[np.
93   newaxis,:,:]))*np.exp(-1j*(k1[:,np.newaxis]-k2[:,np.newaxis]+q1[np.newaxis,:,:])*L/2)
94   -(1+ky**2/(k1[:,np.newaxis])*q1[np.newaxis,:,:])*np.exp(1j*(q1[np.newaxis,:,:])*L/2))
95
96 M_sb_m = Bessel*(Theta_p[np.newaxis,:,:]*sigma1q2[np.newaxis,:,:]*M_sb_m1
97           +Theta_p[np.newaxis,:,:]*qTheta_m[np.newaxis,:,:]*sigma1q1[np.newaxis,:,:]*M_sb_m2
98           +Theta_m[np.newaxis,:,:]*qTheta_p[np.newaxis,:,:]*sigma2q1[:,np.newaxis]*M_sb_m3)
99
100 return M_sa_m, M_sa_p, M_sb_m, M_sb_p
101
102 def M_R(k1, k2, L, sigma_1k1, sigma_1k2):
103     return np.diag(2*sigma_1k1*np.exp(-1j*k1*L/2))
104
105 def M_C(k1, k2, q1, q2, L, Bessel, pTheta, mTheta, qpTheta, qmTheta, sigma1k1, sigma1k2, sigma1q1,
106   sigma1q2, sigma2q1, sigma2q2):
107     M_c_A_m = Bessel/sigma1k2[:,np.newaxis]*(mTheta[np.newaxis,:,:]*qpTheta[np.newaxis,:,:]*sigma2q1[np.
108   .newaxis,:,:]*np.exp(-1j*(k2[:,np.newaxis]-q1[np.newaxis,:,:])*L/2)
109   +pTheta[np.newaxis,:,:]*(qmTheta[np.newaxis,:,:]*sigma1q1[np.newaxis,:,:]*np.exp(-1j*(k2
110   [:,np.newaxis]+q1[np.newaxis,:,:])*L/2)
111   +sigma1q2[np.newaxis,:,:]*np.exp(-1j*(k2[:,np.newaxis]-q2[np.
112   newaxis,:,:])*L/2)))
113
114 M_c_A_p = Bessel/sigma1k2[:,np.newaxis]*(pTheta[np.newaxis,:,:]*qpTheta[np.newaxis,:,:]*sigma1q1[np.
115   .newaxis,:,:]*np.exp(-1j*(k2[:,np.newaxis]+q1[np.newaxis,:,:])*L/2)
116   +mTheta[np.newaxis,:,:]*(qmTheta[np.newaxis,:,:]*sigma2q1[np.newaxis,:,:]*np.exp(-1j*(k1
117   [:,np.newaxis]-q1[np.newaxis,:,:])*L/2)
118   +sigma2q2[np.newaxis,:,:]*np.exp(-1j*(k2[:,np.newaxis]+q2[np.
119   newaxis,:,:])*L/2)))
120
121 M_c_B_m = Bessel/sigma1k1[:,np.newaxis]*(mTheta[np.newaxis,:,:]*qpTheta[np.newaxis,:,:]*sigma2q1[np.
122   .newaxis,:,:]*np.exp(-1j*(k1[:,np.newaxis]+q1[np.newaxis,:,:])*L/2)
123   +pTheta[np.newaxis,:,:]*(qmTheta[np.newaxis,:,:]*sigma1q1[np.newaxis,:,:]*np.exp(-1j*(k1
124   [:,np.newaxis]-q1[np.newaxis,:,:])*L/2)
125   +sigma1q2[np.newaxis,:,:]*np.exp(-1j*(k1[:,np.newaxis]-q2[np.
126   newaxis,:,:])*L/2)))
127
128 return M_c_A_m, M_c_A_p, M_c_B_m, M_c_B_p
129
130 def M_I(k1, k2, L, sigma1k1, sigma1k2):
131     M_i_A = np.diag(sigma1k1*np.exp(-1j*(k1+k2)*L/2)/sigma1k2)
132     M_i_B = np.diag(sigma1k2*np.exp(-1j*(k1+k2)*L/2)/sigma1k1)
133
134 return M_i_A, M_i_B
135
136 def Matrix(M_c_A_m, M_c_A_p, M_c_B_m, M_c_B_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i_A, M_i_B,
137   k1, k2):
138
139 # Produce the inverses
140 M_sa_p_inv = np.linalg.inv(M_sa_p)
141 M_sa_m_inv = np.linalg.inv(M_sa_m)

```

```

130 M_sb_p_inv = np.linalg.inv(M_sb_p)
131 M_sb_m_inv = np.linalg.inv(M_sb_m)
132
133 # Combine the inverses accordingly
134 pM_sa = M_sa_p_inv+M_sa_m_inv
135 mM_sa = M_sa_p_inv-M_sa_m_inv
136 pM_sb = M_sb_p_inv+M_sb_m_inv
137 mM_sb = M_sb_p_inv-M_sb_m_inv
138
139 # Compute the normalizing factor (Re(kn)/Re(km))^-1/2
140 x1 = np.where(np.imag(k1) != 0, 0, k1)[:,np.newaxis]
141 x2 = np.where(np.imag(k2) != 0, 0, k1)[np.newaxis,:]
142
143 norm = np.where(x2 == 0, 0, np.sqrt(x1/x2))
144
145 # Combine other stuff
146 appmm = np.matmul(M_sb_p_inv,M_sa_p)+np.matmul(M_sb_m_inv, M_sa_m)
147 bppmm = np.matmul(M_sa_p_inv,M_sb_p)+np.matmul(M_sa_m_inv, M_sb_m)
148
149
150 # Get the inverses
151 appmm_inv = np.linalg.inv(appmm)
152 bppmm_inv = np.linalg.inv(bppmm)
153
154 # Compute parts of final
155
156 aA = np.matmul(np.matmul(appmm_inv, pM_sb), M_r)
157 aB = np.matmul(np.matmul(appmm_inv, mM_sb), M_r)
158 bA = np.matmul(np.matmul(bppmm_inv, mM_sa), M_r)
159 bB = np.matmul(np.matmul(bppmm_inv, pM_sa), M_r)
160
161 # Compute the final reflection and transmission matrices
162 m_AA = norm*(np.matmul(M_c_A_p, aA)+np.matmul(M_c_A_m, aB)-M_i_A) #Reflection from left
163 m_AB = norm*(np.matmul(M_c_A_p, aB)+np.matmul(M_c_A_m, bB)) #Transmission to left
164 m_BA = norm*(np.matmul(M_c_B_p, aA)+np.matmul(M_c_B_m, bA)) #Transmission to right
165 m_BB = norm*(np.matmul(M_c_B_p, aB)+np.matmul(M_c_B_m, bB)-M_i_B) #Reflection from right
166 return m_AA, m_AB, m_BA, m_BB
167
168 def altfermi(n, hw, E, E_F): # make into a matrix with 0 and 1 to multiply the entire matrix M_nmp
169     return 0.5*(np.sign(E-E_F+n[:, np.newaxis]*hw)-np.sign(E-E_F+n[np.newaxis, :]*hw))**2
170
171 def herm(A):
172     return np.transpose(np.conjugate(A))
173
174 def shot_noise(N, F, sE_LR, sE_LL):
175     sELR = sE_LR[N:, N:]
176     sELL = sE_LL[N:, N:]
177
178     M_LLRR = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis, :]), (F*np.
179     matmul(herm(sELR),sELR)))
180     M_LLRL = np.matmul(np.matmul(herm(sELR)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis, :]), (F*np.
181     matmul(herm(sELL),sELR)))
182     M_LLLR = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELR[0, :][np.newaxis, :]), (F*np.
183     matmul(herm(sELR),sELL)))
184     M_LLLL = np.matmul(np.matmul(herm(sELL)[:, 0][:, np.newaxis], sELL[0, :][np.newaxis, :]), (F*np.
185     matmul(herm(sELL),sELL)))
186
186     M_nmp = M_LLRR+M_LLRL+M_LLLR+M_LLLL
187     return np.trace(M_nmp)
188
189 def Transmission(s_RL, m, N):
190     return np.sum(np.absolute(s_RL[m+N, N:])**2)
191
192 def dxE(A,E_F):
193     return np.diff(A)/np.diff(E_F)
194
195 def main():
196     V_0 = -10 #meV
197     V_1 = 1 #meV
198     mev = 1.602*10**(-22) # mev to joule converter

```

```

196 mu = 0.001*const.m_e
197 hw = 4 #meV
198 L = 3000*(10**(-10))
199 N = 2
200 n = np.arange(-N,N+1)
201 m = np.arange(N+1)
202 Bessel = spec.jv(n[:,np.newaxis]-n[np.newaxis,:], V_1/hw) # The bessel function of the first
kind, over all index combinations
203 E_F = np.linspace(11.5, 12, 100) # Incoming energy in meV
204 fig1, ax1 = plt.subplots(3,1, sharex=True)
205 fig1.subplots_adjust(hspace=0)
206 kys = [0.00095]#, 0.00096, 0.00097]
207 colors = ['blue', 'green', 'red']
208 linestyles = ['-', '--', '-.']
209 Tauy = np.array([0,1,2])*10**4
210 Taux = [0]#np.array([0, 100, 1000])
211 T = [[[[[ for z in range(len(Taux))] for y in range(len(Tauy))] for x in range(len(kys))]]]
212 R = [[[[[ for z in range(len(Taux))] for y in range(len(Tauy))] for x in range(len(kys))]]]
213 S_LL = [[[[[ for z in range(len(Taux))] for y in range(len(Tauy))] for x in range(len(kys))]]]
214 dS_LL = [[[[[ for z in range(len(Taux))] for y in range(len(Tauy))] for x in range(len(kys))]]]
215 for i, ky in enumerate(kys):
216     ky *= 10**(10)
217     for j, tauy in enumerate(Tauy):
218         for l, taux in enumerate(Taux):
219             tau = taux*mu/const.hbar
220
221             for E_0 in E_F:
222                 k1, k2 = free(n, E_0, mu, hw, ky, tau, tauy, mev)
223                 q1, q2, Theta_p, Theta_m, qTheta_p, qTheta_m = barrier(n, E_0, mu, V_0, hw, ky,
tau, tauy, mev)
224                 sigma_1k1 = varsigma1(ky, k1)
225                 sigma_1k2 = varsigma1(ky, k2)
226                 sigma_1q1 = varsigma1(ky, q1)
227                 sigma_1q2 = varsigma1(ky, q2)
228                 sigma_2q1 = varsigma2(ky, q1)
229                 sigma_2q2 = varsigma2(ky, q2)
230
231                 (M_c_A_m, M_c_A_p, M_c_B_m, M_c_B_p, M_r, M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i_A
,
232 M_i_B) = Matrices(k1, k2, q1, q2, ky, L, m, hw, Bessel, Theta_p, Theta_m,
qTheta_p,
233 qTheta_m, sigma_1k1, sigma_1k2, sigma_1q1, sigma_1q2,
sigma_2q1, sigma_2q2)
234
235                 s_LL, s_LR, s_RL, s_RR = Matrix(M_c_A_m, M_c_A_p, M_c_B_m, M_c_B_p, M_r,
236 M_sa_m, M_sa_p, M_sb_m, M_sb_p, M_i_A, M_i_B,
k1, k2)
237
238                 T[i][j][1].append(Transmission(s_RL, 0, N))
239                 #R[i][j][1].append(Transmission(s_LL, 0, N))
240
241
242                 S_ll = 0
243                 Emax = E_0
244                 Esteps = 300
245                 E = np.linspace(3.5, Emax, Esteps)
246                 dE = Emax/Esteps
247                 for e in E:
248                     F = altfermi(m, hw, e, E_0)
249
250                     k1E, k2E = free(n, e, mu, hw, ky, tau, tauy, mev)
251                     q1E, q2E, Theta_pE, Theta_mE, qTheta_pE, qTheta_mE = barrier(n, e, mu, V_0,
hw, ky, tau, tauy, mev)
252                     sigma_1k1E = varsigma1(ky, k1E)
253                     sigma_1k2E = varsigma1(ky, k2E)
254                     sigma_1q1E = varsigma1(ky, q1E)
255                     sigma_1q2E = varsigma1(ky, q2E)
256                     sigma_2q1E = varsigma2(ky, q1E)
257                     sigma_2q2E = varsigma2(ky, q2E)

```

```

259
260             (M_c_A_mE, M_c_A_pE, M_c_B_mE, M_c_B_pE, M_rE, M_sa_mE, M_sa_pE, M_sb_mE,
261             M_sb_pE, M_i_AE,
262             M_i_BE) = Matrices(k1E, k2E, q1E, q2E, ky, L, m, hw, Bessel, Theta_pE,
263             Theta_mE, qTheta_pE,
264             sigma_1q2E, sigma_2q1E, sigma_2q2E)
265
266             sE_LL, sE_LR, sE_RL, sE_RR = Matrix(M_c_A_mE, M_c_A_pE, M_c_B_mE, M_c_B_pE,
267             M_rE,
268             M_sa_mE, M_sa_pE, M_sb_mE, M_sb_pE, M_i_AE,
269             M_i_BE, k1E, k2E)
270             S_ll += shot_noise(N, F, sE_LR, sE_LL)
271
272             S_LL[i][j][1].append(S_ll*dE)
273             dS_LL[i][j][1] = ddE(S_LL[i][j][1], E_F)
274
275             ax1[0].plot(E_F, T[i][j][1], color = colors[1], linestyle = linestyles[i])
276             #ax1[0].plot(E_F, R[i][j][1], color = colors[1], linestyle = linestyles[i+1])
277
278             ax1[1].plot(E_F, S_LL[i][j][1], c = colors[1], label=fr'$\tau_y=\${tauy} m/s')
279             ax1[0].set_ylabel('$T$')
280             ax1[1].set_ylabel('$\mathcal{N}$')
281             ax1[2].set_ylabel('$d\mathcal{N}/dE_F$')
282             ax1[1].legend()
283             ax1[2].set_xlabel('$E_F$ [meV]')
284             plt.show()
285
286 main()

```