The Effect of a Magnetic Field on a S/FE/S Junction

Keane Ramdin (4850955)

Delft, July 18, 2023

Bachelor Thesis
TU Delft, Faculty of Applied Sciences
BSc Applied Physics
TU Delft, Faculty of EEMCS
BSc Applied Mathematics

Supervisors:
Dr. M.N. Ali
Dr. Y.M. Blanter
Dr. P.M. Visser
Dr. T.W.C. Vroegrijk

Abstract

There has been interesting research on the superconducting diode recently. A possible structure for this is the Superconductor/Ferroelectric/Superconductor (S/FE/S) Josephson junction. Our goal is to simulate a S/FE/S junction with a magnetic field and see if this leads to a superconducting diode effect. First this junction and its properties are studied through a numerical simulation. Then a magnetic field is applied and we look at how this affects the junction. For our simulation we use the Velocity Verlet method. We start by simulating a normal RCSJ. The results agree with what we expect from the theory, however there is small but noticeable error which we attribute to the numerical method. Next a S/FE/S junction is simulated. The polarisation of the ferroelectric only affects the RCSJ when it oscillates at a frequency which is close to the plasma frequency. If we apply a magnetic field we see small gaps for the retrapping current however these are close to the numerical error thus they are to small too be detected by our simulation.

Contents

1. Introduction

In the modern era, electronic devices have become ubiquitous, making it difficult to envision our lives without them. From mobile phones to household lighting, the influence of these devices is evident in every aspect of our daily routines. These electric devices consist of electric components and we call a system om these components an electric circuit. The electric circuit of a device can vary greatly, leading to different devices with different functions. However there are only a limited amount of different electrical components and no matter how complex the electrical circuit may seem it is still built of these electrical components. This means that the technology depends on these components. Therefore if we discover new components this may lead to new electronic devices. Such was the case for the semiconducting diode. This discovery paved the way for much of our current technology. So much so that semiconductors are in almost all of the today's electronic devices such as computers, mobile phones, refrigerators and televisions.

Recently there has been a lot of interest in the superconducting diode. This should come as no surprise considering the discovery of the semiconducting diode has had a major impact on technology. Super conducting diodes are not quite the same as semiconductor diodes. While the latter only allow current to flow in one direction the former allows current to flow in both directions but one of these directions has electrical resistance while the other does not. Thus the superconducting diode is superconducting in only one direction. Possible applications of superconducting diodes include superconducting electronics and quantum information and communication technology [1, 2].

The superconducting diode effect has already been observed in both junction free superconductors [3, 4] and Josephson Junctions [5, 6]. However the topic is still relatively new and there is till a lot to be discovered. Inspired by recent developments we will study a superconducting/ferroelectric/superconductor (SFES) Josephson junction and the effect of a magnetic field on this junction. This is done through a numerical simulation using the Velocity Verlet method.

First we will discuss the theory needed to understand the SFES junction. In chapter 2 we explain the theory of Josephson junctions. In chapter 3 the theory of ferroelectricity is treated. Chapter 4 combines the knowledge of these two chapters and applies it to the SFES junction. Chapter 5 gives a explanation of the Velocity Verlet method and how we can apply this to our problem. In chapter 6 we will show the results and discuss them. In chapter 7 we state our conclusions. In appendix A a table of the used parameters is given. In appendix B the python code used for the simulations is given.

2. Theory of Josephson Junctions

2.1. The Josephson effect

A Josephson junction is a thin gap of two superconductors with a "weak link" in between. The effects of superconductivity are extended inside this "weak link".

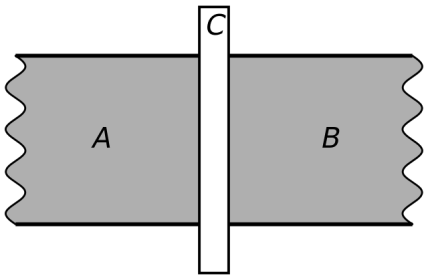

Figure 1: An schematic of a Josephson junction. In this figure A and B are superconductors and C thin insulating barrier. Image taken from [7]

In this junction a nonzero current can flow while the voltage is zero. This is called the DC Josephson effect and the super current is given by

$$I_s = I_c \sin(\Delta\varphi), \tag{1}$$

where I_s is the super current, I_c is the maximal zero resistance current and $\Delta\varphi$ is the phase difference between the two superconducting electrodes.

If a voltage is applied to the junction we get the AC Josephson effect : the phase changes in time according to

$$\frac{d\Delta\varphi}{dt} = \frac{2eV}{\hbar}, \tag{2}$$

where V is the voltage, e is the electron charge and $\hbar$ is Planck's reduced constant. This means that if we apply a constant voltage to a Josephson junction the super current will oscillate with frequency

$$\nu = \frac{2eV}{\hbar} \tag{3}$$

2.2. RCSJ model

When the applied current I is less than the critical current I_c the junction is superconducting and the zero voltage current is given by equation (1). However if the applied current is larger than the critical current the junction has a capacitance and resistance. To describe this we use the Resistively and Capacitively Shunted Juntion (RCSJ) model. In this model the physical junction is represented by an ideal junction (one without resistance and capacitance) shunted by a resistor and capacitance. This is illustrated in figure 2.

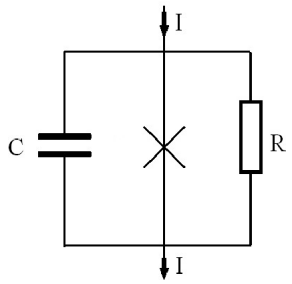

Figure 2: The Resistively and Capacitively Shunted Junction (RCSJ) model circuit. The cross in the middle represents an ideal Josephson junction. On the left side is a capacitor with capacitance C and on the right side is a resistor with resistance R. Image taken from [8]

The applied current splits into three different currents. One going through the capacitor which we shall call I_C, one going through the resistor which we will call I_R, and one going through the ideal Josephson junction which we shall call I_s. The sum of the currents in the branches of figure 2 should be equal to the applied current I. This gives:

$$I = I_s + I_R + I_C. \tag{4}$$

Using the current voltage relations of a capacitor $I_C = C\frac{dV}{dt}$ and a resistor $I_R = \frac{V}{R}$ together with the current through an ideal Josephson junction given by equation (1) we obtain the following:

$$I = I_c \sin(\varphi) + C\frac{dV}{dt} + \frac{V}{R} \tag{5}$$

We introduce the following variables

$$\omega_p = \left(\frac{2eI_c}{\hbar C}\right)^{\frac{1}{2}}, \tag{6}$$

$$\tau = \omega_p t, \tag{7}$$

$$Q = \omega_p RC. \tag{8}$$

Here, ω_p is the plasma frequency which will be explained later when we treat the "tilted washboard potential". $\frac{1}{Q}$ is the damping factor. Using these variables and equation (2), we can write equation (5) as:

$$\frac{d^2\Delta\varphi}{d\tau^2} + \frac{1}{Q}\frac{d\Delta\varphi}{d\tau} + \sin(\Delta\varphi) = \frac{I}{I_c}. \tag{9}$$

If $I < I_c$, then there is no voltage in the junction and thus we can use equation (1) to obtain $\Delta\varphi = \sin^{-1}\left(\frac{I}{I_c}\right)$. If $I > I_c$ we distinguish two regimes, one with high damping and one with low damping. If $Q << 1$, the damping is high and the second term on the left hand side of equation (13) will dominate. In this case the differential equation reduces to a first order differential equation:

$$\frac{d\Delta\varphi}{dt} = \frac{2eI_cR}{\hbar}\left(\frac{I}{I_c} - \sin(\Delta\varphi)\right). \tag{10}$$

This is called a overdamped junction. Note that $\Delta\varphi$ is a periodic function. If we integrate (10) we can find the period T of $\Delta\varphi$, then using (3) we can find V with

$$\nu \equiv \frac{2\pi}{T} = \frac{2eV}{\hbar}. \tag{11}$$

We then get [9]

$$V = R(I^2 - I_c^2)^{\frac{1}{2}}. \tag{12}$$

This is illustrated in figure 3.

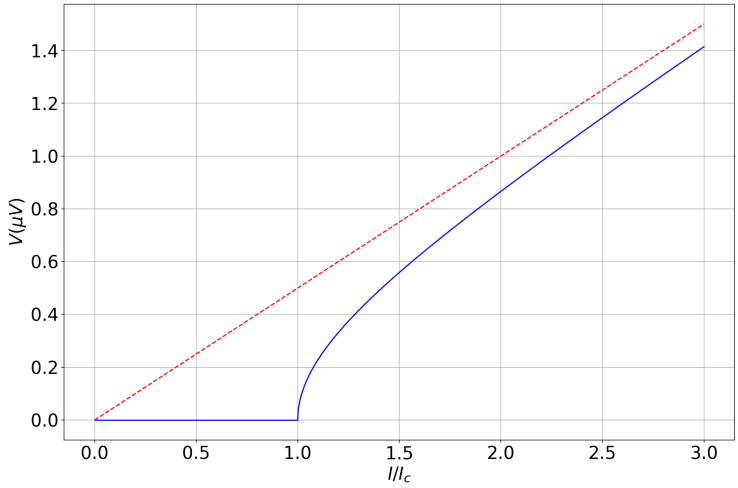

Figure 3: The voltage is plotted against the current for a overdamped RCSJ (the solid blue line). The red dotted line is given by $V = IR$, which is Ohm's law. The voltage is zero when I is smaller than the critical current I_c. When $I \gg I_c$ the curve approaches $V = IR$ asymptotically.

For $I < I_c$ we have $V = 0$ and for $I \gg I_c$ we get Ohm's law: $V = IR$.

If Q is not too small, the damping term will not dominate. The junction is then called underdamped and equation (13) needs to be solved numerically. When we do this we get a I-V curve which is hysteretic. This is illustrated in figure 4.

We start by increasing the current from zero. This is the blue line. As long as the current is smaller than the critical current the voltage is zero. When the current surpasses the critical current the voltage jumps to a finite value and now increases linearly. When we now decrease the current we get the orange line. We note that the current does not jump back to zero at the critical current but at a smaller current. This is called the retrapping current. For negative current the behaviour is the same but now the voltage is negative as indicated by the green and red lines.

One way to understand the behaviour of the RCSJ is through the "tilted washboard potential" [9]. We can write equation (5) as [10]:

$$C\frac{\hbar}{2e}\frac{d^2\Delta\varphi}{dt^2} + \frac{1}{R}\frac{\hbar}{2e}\frac{d\Delta\varphi}{dt} + \frac{\partial U}{\partial\Delta\varphi} = 0, \tag{13}$$

where

$$U = -I_c\cos(\Delta\varphi) - I\Delta\varphi \tag{14}$$

We can thus interpret the phase as a particle in this potential. The potential consists of a cosine term and a linear term. Thus it is a tilted cosine whose slope is equal to I. as shown in figure 5.

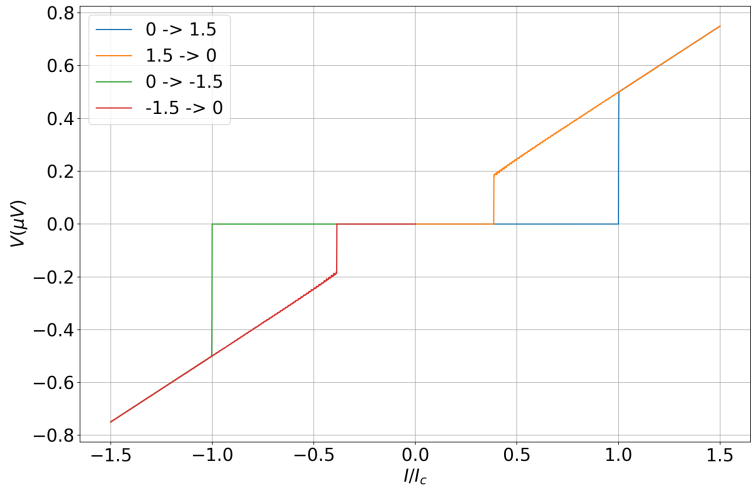

Figure 4: A hysteretic I-V curve. Along the x axis we have the current I divided by the critical current I_c. Along the y axis we have the voltage in μV The blue line represents a current sweep from 0 to 1.5, the orange line represents a current sweep from 1.5 to 0, the green line represents a current sweep from 0 to -1.5 and the red line represents a current sweep from -1.5 to 0. As we increase the current from 0 the voltage remains zero as long as the current is smaller than the critical current. When the current surpasses the critical current the voltage jumps to a finite value and now increases linearly. When decreasing the current back to 0 the voltage does not jump back to 0 at the critical current but at a smaller current. For negative current we have the same behaviour except now the voltage is negative.

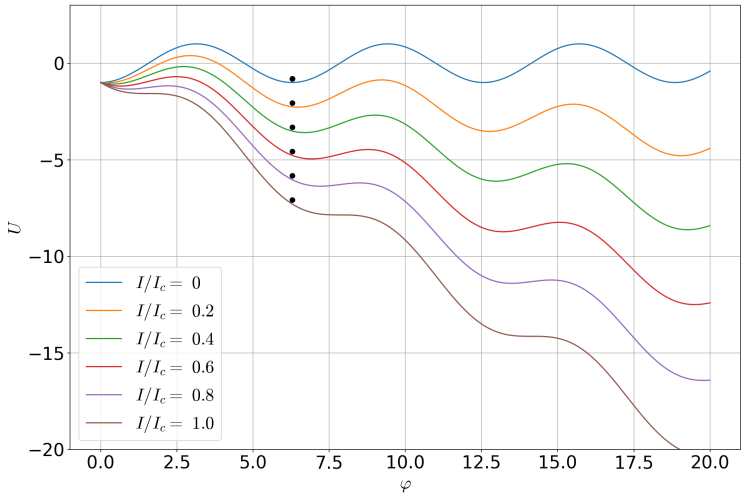

Figure 5: The tilted washboard potential is shown for different currents between 0 and I_c. For $I = 0$ the potential is a cosine. A particle in this potential is trapped around a minimum. As I is increased from zero the potential starts to tilt downward. As this happens the right side of the barrier containing the particle becomes smaller. When I reaches the critical current there are no more barriers and the particle can move freely.

As seen in the figure for zero current the potential and the phase can be interpreted as a particle trapped around one of the minima. If we increase the current the potential tilts downward. If the tilt is big enough there are no more minima and the particle can roll downwards. The minima vanish for $I \geq I_c$.

We can use the "tilted washboard potential" to understand the RCSJ graph in figure 4. If we increase I from zero the potential stars as a normal cosine and tilts downward with increasing I. The particle is prevented from rolling down due to the barrier of the cosine and oscillates in a well. The frequency at which it oscillates is the plasma frequency given by equation (6). When I reaches the critical current there are no more barriers and the particle can roll down. This causes a voltage jump. When we decrease I the particle now has a momentum and thus with light dampin the particle can still move over small barriers. This explains why the retrapping current is smaller then critical current. If the current is negative the potential tilts upwards and the particle moves in the other direction. This causes a negative voltage.The rest of the process is the same as for positive current.

2.3. Josephson Junction with a Static Magnetic Field

We now apply a static magnetic field perpendicular to the junction. We will see that the magnetic field has an influence on the phase difference and thus from (1) also on the current. Here we will take a look at a magnetic field applied to a rectangular Josephson junction and the resulting effects.

The theory of superconductivity tells us that magnetic fields are expelled from superconductors. This is known as the Meissner effect. However the magnetic field penetrates a tiny distance into a superconductor [9]. This depth is called the penetration depth.

We apply a magnetic field **B** perpendicular to a Josephson junction and choose a flux surface as indicated with the dotted line in figure 6.

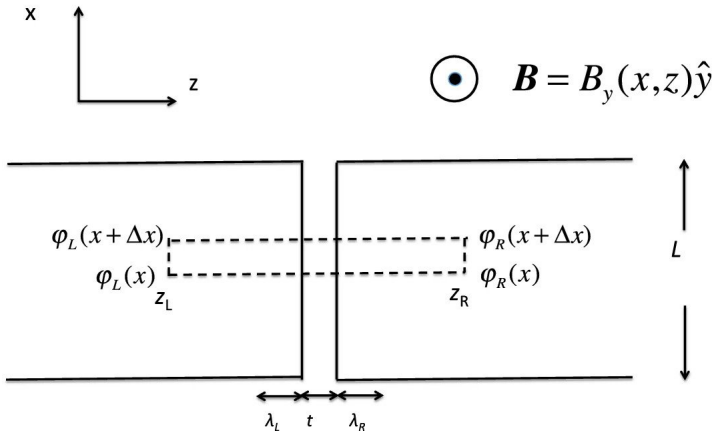

Figure 6: A rectangular Josephson junction with a magnetic field **B** applied in the y direction. The width of the gap between the superconductors is t. λ_L and λ_R are the left and right penetration depth of the magnetic field respectively. L is the height of the junction. $\varphi_L(x)$ and $\varphi_R(x)$ indicate the phase of the left and right superconductor at x respectively. The dotted line is the contour along which we integrate the phase gradient. z_L and z_R indicate the left and right edge of the contour respectively. Image taken from [11].

The magnetic field **B** is in the y direction, $\mathbf{B} = B_y(x,z)\hat{y}$. The junction has height L and the thickness of the barrier is t. The magnetic field penetrates the left and right superconductor with depth λ_L and λ_R respectively. We define

$$d = \lambda_L + t + \lambda_R. \tag{15}$$

d represents the width of the region where the magnetic field is non zero. The phase in the left and right superconductors are given by φ_L and φ_R respectively.

We also define

$$\Phi_0 \equiv \frac{h}{2e} = \frac{2\pi\hbar}{2e} \tag{16}$$

Φ_0 is known as the magnetic flux quantum.
The gauge - invariant phase difference in a Josephson junction γ is given by [9]

$$\gamma \equiv \Delta\varphi - \frac{2\pi}{\Phi_0}\int \mathbf{A}\cdot d\mathbf{s} \tag{17}$$

where $\mathbf{A}$ is the magnetic vector potential defined by $\mathbf{B} = \nabla \times \mathbf{A}$ and $\Delta\varphi$ is the phase difference across the gap. If there is no magnetic field the gauge-invariant phase difference is the same as the phase difference.
If there is a magnetic field we get:

$$\gamma(x) = \varphi_L(x) - \varphi_R(x) - \frac{2\pi}{\Phi_0}\int A dz, \tag{18}$$

where we integrate from z_l to z_r. The phase gradient is given by [11]

$$\nabla\varphi = \frac{2e}{\hbar}\Big(\frac{m\mathbf{J}}{2e^2\rho} + \mathbf{A}\Big) \tag{19}$$

In our junction we can neglect the first term because **J** is perpendicular in the parts of the contour parallel to the x axis and negligibly deep in the parts of the contour parallel to the z axis. We then get

$$\nabla\varphi = \frac{2e}{\hbar}\mathbf{A} = \frac{2\pi}{\Phi_0}\mathbf{A} \tag{20}$$

If we now go one trip along the dotted line in figure 6 we land at the same spot where the phase must be the same. Thus we must have a phase gain of 0. So we must have [12]

$$\oint \nabla\varphi \cdot d\mathbf{l} = \frac{2\pi}{\Phi_0}\oint \mathbf{A}\cdot d\mathbf{l} + \Delta\varphi(x) - \Delta\varphi(x+\Delta x) = 0 \tag{21}$$

Now we assume **B** is constant. Using the definition of **A** gives:

$$\Delta\varphi(x+\Delta x) - \Delta\varphi(x) = \frac{2\pi}{\Phi_0} B d \Delta x \tag{22}$$

Dividing by Δx and taking the limit $\Delta x \to 0$ we get

$$\frac{\partial\Delta\varphi}{\partial x} = \frac{2\pi B d}{\Phi_0} \tag{23}$$

This tells us that the phase difference across the gap varies in the junction along the x axis. Namely

$$\Delta\varphi(x) = \Delta\varphi_0 + \frac{2\pi B d}{\Phi_0}x \tag{24}$$

Equation (1) then says that this causes a non uniform current along the x axis Since the current in not uniform in the junction we need to look at the current density, which is the current per unit area. Using (1) and (23) we get

$$J = J_{c0}\sin\Big(\Delta\varphi_0 + \frac{2\pi B d}{\Phi_0}x\Big) \tag{25}$$

If we integrate (25) over the junction we get the critical current as a function of the flux $\Phi = BdL$

$$I_c(\Phi) = I_{c0}\left|\frac{\sin\left(\pi\frac{\Phi}{\Phi_0}\right)}{\pi\frac{\Phi}{\Phi_0}}\right| \tag{26}$$

Where $I_{c0} = J_{c0}BdL$. This is called the Fraunhofer pattern. This is exactly the diffraction pattern we get when sending light through a single slit. This pattern is illustrated in figure 7

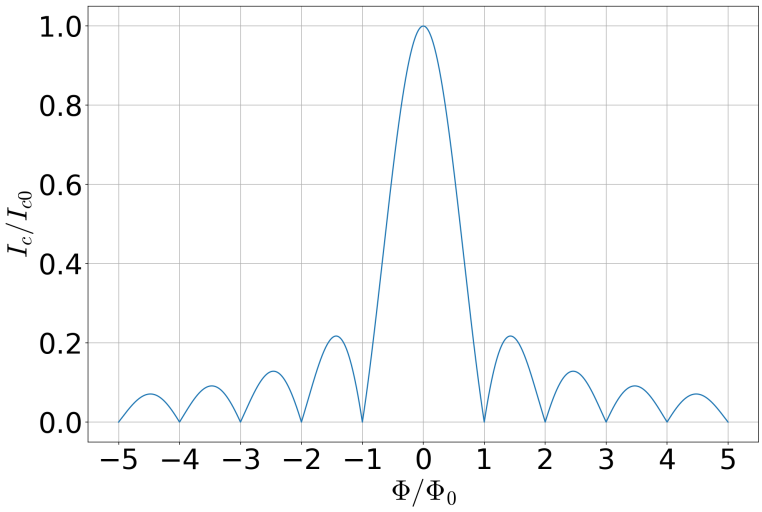

Figure 7: The Fraunhofer pattern is shown. The x axis is the flux divided by the magnetic flux quantum Φ_0. The y axis is the critical current divided by the critical current at zero flux. When there is no flux the critical current is maximum. When the flux is an integer multiple of the magnetic flux quantum the critical current is 0.

The RSCJ $I - V$ graph is shown for different flux values in figure 8

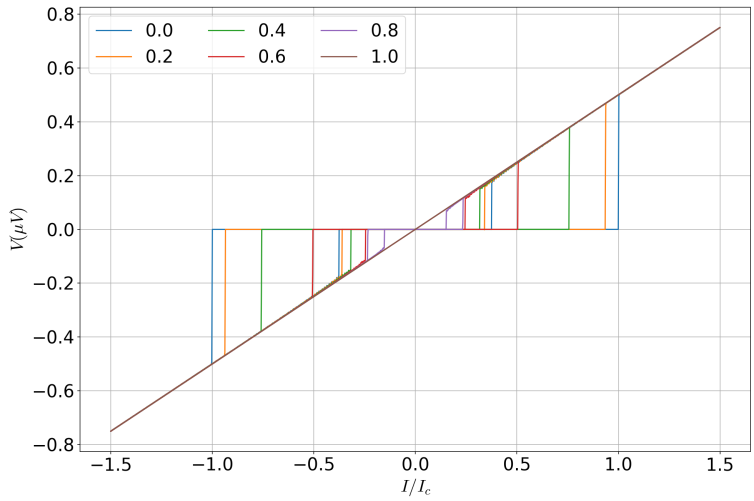

Figure 8: The graph of a RCSJ for different values of the flux in units of the magnetic flux quantum.Φ_0. The x axis is the current in units of the critical current. The y axis is the voltage. Note that for zero flux the graph is the same as figure 4 as expected. When the flux is equal to the the magnetic flux quantum the graphs is just a straight line so the RCSJ behaves according to Ohm's law. This is what we expect as according to equation (26) the critical current is 0 when the flux is a integer multiple of the magnetic flux quantum.

3. Theory of Ferroelectrics

3.1. Landau Devonshire theory

A Josephson junction is a electrical component with a ,current-voltage relation, just like a resistor or a capacitor. The current-voltage relation of resistors and capacitors is determined by the the resistance and capacitance respectively. These are dependent on the material and the physical form of the resistor and capacitor. Similarly, in a Josephson junction the shape and material affect the current-voltage relation. Here the material can be the superconductors themselves or the material in the gap. An interesting idea is to put a ferroelectric material in the gap of a Josephson junction. A ferroelectric material has an intrinsic electrical polarisation, which means that the material has a electrical polarisation even in the absence of a electric field. This is in contrast to dielectric materials which only have a electrical polarisation in the presence of an electric field. With a electrical polarisation the junction is no longer symmetric as current in one direction aligns with the polarisation while current in the other direction is opposite to the polarisation. This asymmetry may lead to the current-voltage relation being dependent on the direction of the current which may lead to a superconducting diode effect. The intrinsic polarisation in a ferroelectric is caused by the displacement of atoms in the structure which causes a net polarisation in the material.

Ferroelectrics are defined as materials with a spontaneous polarisation that can be reoriented with an applied electric field [13]. We will describe the theory of ferroelectrics in this section. In order to do this we will use Landau-Devonshire theory which uses the Landau theory of phase transitions to explain ferroelectricity [14].

Landau theory characterizes a phase transition in terms of a order parameter. This order parameter is 0 when there is high symmetry and has a finite value when the symmetry is broken. For ferroelectrics the order parameter is the polarisation P. The free energy can be expressed as a power series of the order parameter P. In this expansion, ignoring the effect of an external electric field E, we keep only terms which are symmetric. So we assume $F(-P) = F(P)$ [14]. This gives :

$$F(P) = \frac{a}{2}P^2 + \frac{b}{4}P^4 - EP \tag{27}$$

where we have left out sixth order and higher terms. We have :

$$a = \frac{1}{\chi}, \tag{28}$$

where χ is the linear dielectric susceptibility given by [13] $\varepsilon_r = 1 + \chi$, where ε_r is the relative electric permittivity. For temperatures close to the Curie temperature :

$$a = a_0(T - T_c), \tag{29}$$

where a_0 is positive for ferroelectrics, T is the temperature and T_c is the Curie temperature.

We assume that we have a second order transition which at $T = T_c$ occurs when $b > 0$. Second order transitions have continuous first derivative but discontinuous second derivative [15]. The spontaneous polarisation P_s are the equilibrium points of (27) in the absence of an electric field. In order to find these we take the first derivative of the free energy and set it equal to zero, we then get:

$$0 = aP + bP^3 - E. \tag{30}$$

Setting $E = 0$ we find for the spontaneous polarisation :

$$P_s = \pm\sqrt{\frac{-a}{b}}. \tag{31}$$

In equation (31) we have two unknown variables P_s and b. One of these must be determined experimentally and the other can then be obtained- from 31.

For ferroelectric materials $a < 0$ and $a_0 > 0$ [14]. If we look at equation (29) we see that materials are in the ferroelectric state when $T < T_c$. This means that at T_c we have a transition from a paraelectric to a ferroelectric state. This is illustrated in figure 9.

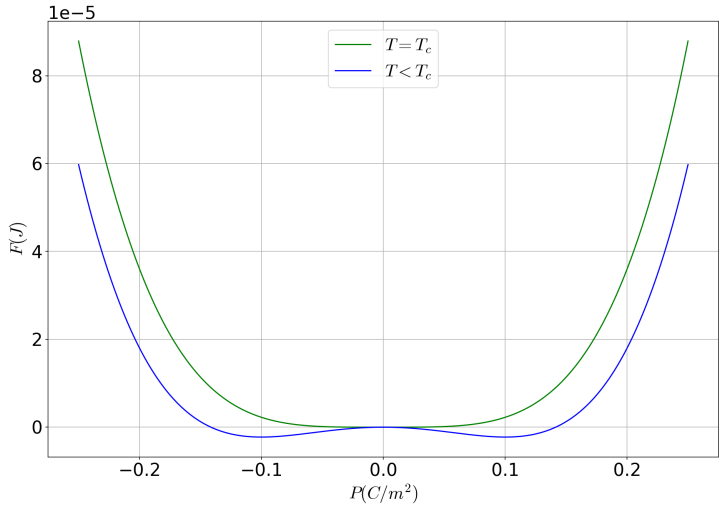

Figure 9: Transition from a paraelectric state, the green line at $T = T_c$, to a ferroelectric state, the blue line at $T < T_c$. We see that the free energy goes form having one minimum at zero to having to minima at $\pm P_s$.

We now apply an electric field. According to equation (27) the free energy depends linearly on the electric field. This means the graph can be seen as a fourth order polynomial plus a linear term. Visually we can interpret this as the electric field giving a tilt to the graph in figure 9. For positive field strength $E > 0$ this is shown in figure 10.

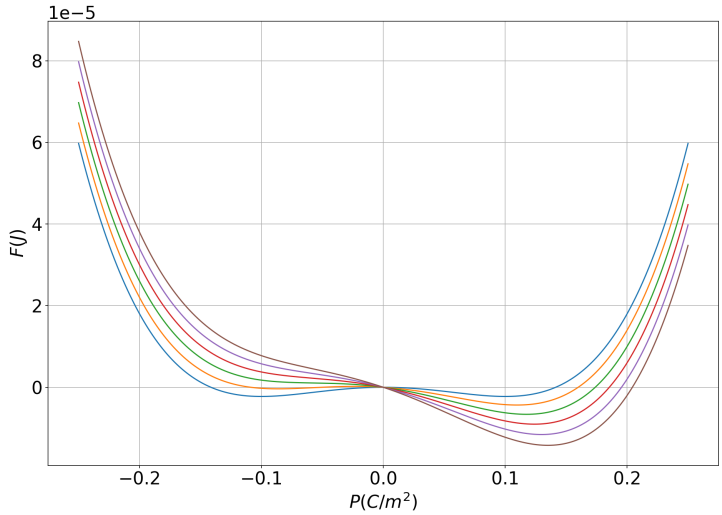

Figure 10: The polarisation free energy for different voltages. The blue line corresponds to zero voltage and the brown line corresponds to the highest voltage. The voltage applied is positive which causes the graph to tilt downward.

We observe that at some point the free energy only has one minimum. Thus there is some voltage for which the system goes from having two minima to one. In order to find this critical voltage we note that at the turning point a minimum becomes a saddle point. Hence we need to calculate the first and second order derivative and set them equal to zero. This gives for the critical voltage:

$$\begin{cases} 0 = F'(P) = aP + bP^3 - V \\ 0 = F''(P) = a + 3bP^2 \end{cases} . \tag{32}$$

Solving for P and V gives:

$$P_{0_\pm} = \pm\sqrt{\frac{-a}{3b}} \tag{33}$$

and

$$V_{c_\mp} = \mp\frac{2b}{3\sqrt{3}}P_0^3. \tag{34}$$

Note that since $a < 0$ and $b > 0$ the root is real. In equation (34) there are two critical voltages, one positive and one negative. These correspond to different tilts of the graph. When we have a positive voltage the graph tilts downward, as shown in figure 10, and the saddle point occurs at negative polarisation. When the voltage is positive the graph tilts upwards and the saddle point occurs at positive polarisation. In figure 11 the free energy is shown for zero voltage (green), positive critical voltage(red) and negative critical voltage (blue).

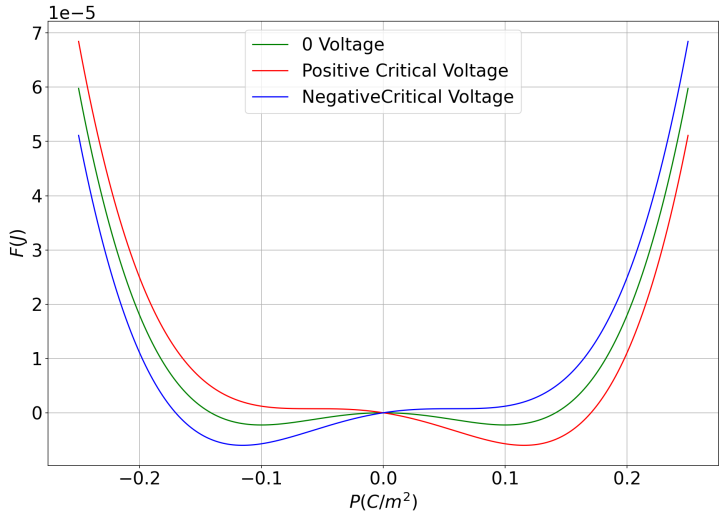

Figure 11: The free energy as a function of the polarisation for zero voltage, positive critical voltage and negative critical voltage. The green line corresponds to zero voltage, the red line corresponds to positive critical voltage and the blue line corresponds to negative critical voltage. For zero voltage we have a minimum at $P = \pm P_s$. For positive critical voltage V_{c+} we have a saddle point at P_{0-}. For negative V_{c-} we have a saddle point at P_{0+}.

3.2. Dynamics of Polarisation

We now look at the dynamical behaviour of the polarization. For this we use the free energy as the potential in which the polarisation finds itself [16]. We then use Newton's equation of motion with a drag term to obtain

$$m\frac{d^2P}{dt^2} + \gamma_P\frac{dP}{dt} = V - aP - bP^3 \tag{35}$$

where m is the inertia of the polarisation and hence the effective mass. γ_P is the damping factor. Equation (35) resembles an oscillator with a damping factor γ_P, mass m and a restoring force $-\frac{dF(P)}{dP} = V - aP - bP^3$. If we look at figure 9 we can solve equation (35) when P is in one of the minima $\pm P_s$. For this oscillation we can calculate the frequency. To do this we assume that P is close to one of the minima so

$$P = P_s + \delta P, \tag{36}$$

where δP is small enough so that P is inside the well. With equation (35) and equation (36) and setting $V = 0$ we obtain :

$$m\delta\ddot{P} + \gamma_P \delta\dot{P} = -a(P_s + \delta P) - b(P_s^3 + 3P_s^2\delta P + 3P_s\delta P^2 + \delta P^3). \tag{37}$$

Leaving out all terms with second or higher order δP we get :

$$m\delta\ddot{P} + \gamma_P \delta\dot{P} = -a(P_s + \delta P) - b(P_s^3 + 3P_s^2\delta P). \tag{38}$$

We assume that equation has solution of the form :

$$\delta P = e^{-zt}, \tag{39}$$

where $z = \Gamma + i\omega_{\text{pol}}$ is a complex number. With equation (38) and (39) and using equation (31) for P_s we obtain :

$$mz^2 - \gamma_P z = 2a. \tag{40}$$

Solving for z we get :

$$z = \frac{\gamma_P}{2m} \pm \frac{\sqrt{\gamma_P^2 + 8am}}{2m}. \tag{41}$$

Since a is negative, if $\gamma_P \geq 2\sqrt{-2am}$, z is real and positive. In this case δP decays exponentially. If $\gamma_P < 2\sqrt{-2am}$, z is complex and δP oscillates. We get:

$$z = \frac{\gamma_P}{2m} \pm \frac{i\sqrt{-\gamma_P^2 - 8a}}{2m}. \tag{42}$$

Thus the system has frequency:

$$\omega_{\text{pol}} = \sqrt{\frac{-2a}{m}}\left(\sqrt{1 + \frac{\gamma_P^2}{8ma}}\right), \tag{43}$$

and period

$$T = \pi\sqrt{\frac{m}{-2a}}\left(\sqrt{\frac{1}{1 + \frac{\gamma_P^2}{8ma}}}\right), \tag{44}$$

and damping rate,

$$\Gamma = \frac{\gamma_P}{2m}. \tag{45}$$

The damping time is therefore

$$t_\Gamma = \frac{2m}{\gamma_P}. \tag{46}$$

4. The Superconductor Ferroelectric Superconductor Josephson Junction

We now have everything we need to analyse a superconductor ferroelectric superconductor Josephson junction. In order to describe the effect of the polarisation in a Josephson junction we will ad the term $\frac{dP}{dt}$ to equation (5).[17] We then get:

$$I = I_c \sin(\varphi) + \frac{V}{R} + C\frac{dV}{dt} + \frac{dP}{dt}. \tag{47}$$

We use equations (6), (7) and (8) to get:

$$\frac{d^2\varphi}{d\tau^2} = \frac{I}{I_c} - \sin(\varphi) - \frac{1}{Q}\frac{d\varphi}{d\tau} - \frac{\omega_p}{I_c}\frac{dP}{d\tau} \tag{48}$$

With the polarisation described by equation (35) and equation(2) we get the following system of second order differential equations:

$$\begin{cases} \dfrac{d^2\varphi}{d\tau^2} = \dfrac{I}{I_c} - \sin(\varphi) - \dfrac{1}{Q}\dfrac{d\varphi}{d\tau} - \dfrac{\omega_p}{I_c}\dfrac{dP}{d\tau} \\ \dfrac{d^2P}{d\tau^2} = \dfrac{\hbar}{2em\omega_p}\dfrac{d\varphi}{d\tau} - \dfrac{1}{m\omega_p^2}(aP + bP^3) - \dfrac{\gamma_p}{m\omega_p}\dfrac{dP}{d\tau}. \end{cases} \tag{49}$$

The system (49) can be understood by looking at each of the equations separately. Assume that the polarisation starts in one of equilibrium points of the free energy with zero voltage. Thus the initial polarisation is equal to the one of the spontaneous polarisations. Since it is in a equilibrium point it does not change. We start by increasing the current from zero. As long as the current is smaller than the critical current the junction is in the superconducting state and there is no voltage. When the current surpasses the critical current we get a finite voltage in the junction. This voltage causes a tilt in the free energy as shown in figure 10. This causes the minima to move and so the polarisation is now no longer in one of the equilibria but is slightly displaced. The polarisation will now oscillate around the new equilibrium. Hence $\frac{dP}{d\tau} \neq 0$ and thus this influences the first equation.

5. The Velocity Verlet Method

We solve the differential equations in this report numerically. For this we use the Velocity Verlet method. The velocity Verlet method is used to solve equations of the form

$$\ddot{x}(t) = a(x(t)). \tag{50}$$

To solve this numerically the Velocity Verlet method uses the following:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{51}$$

$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2}\Delta t, \tag{52}$$

where $v(t) = \dot{x}(t)$ is the first derivative of $x(t)$ and $a(t) = \ddot{x}(t)$ is the second derivative of $x(t)$. Applying this method to our system of equations (49) we get :

$$\varphi(t+\Delta t) = \varphi(t) + \dot{\varphi}(t)\Delta t + \frac{1}{2}\ddot{\varphi}(t)\Delta t^2 \tag{53}$$

$$P(t+\Delta t) = P(t) + \dot{P}(t)\Delta t + \frac{1}{2}\ddot{P}(t)\Delta t^2. \tag{54}$$

$$\dot{\varphi}(t+\Delta t) = \dot{\varphi}(t) + \frac{\ddot{\varphi}(t)+\ddot{\varphi}(t+\Delta t)}{2}\Delta t \tag{55}$$

$$\dot{P}(t+\Delta t) = \dot{P}(t) + \frac{\ddot{P}(t)+\ddot{P}(t+\Delta t)}{2}\Delta t. \tag{56}$$

With the Velocity Verlet integrator the idea is to calculate $x(t+\Delta t)$ for a given time t and time step Δt. Since $x(t+\Delta t)$ only depends on data from time t this step is straightforward, just use the values at time t tot calculate $x(t+\Delta t)$. The next step is to calculate $a(t+\Delta t)$ using $x(t+\Delta t)$. Usually when using the Velocity Verlet algorithm $a(t+\Delta t)$ is a function of $x(t+\Delta t)$ only and thus this is also simple. However in our case we have that $a(t+\Delta t)$ depends on $v(t+\Delta t)$ for both the polarisation P and the phase φ, as seen in equations (49). To solve this problem we start by writing out $\ddot{P}(t+\Delta t)$ in equation (56). We then obtain :

$$\dot{P}(t+\Delta t) = \dot{P}(t) + \frac{\ddot{P}(t)}{2}\Delta t + \frac{\Delta t}{2}\left[\frac{\hbar}{2em\omega_p}\dot{\varphi}(t+\Delta t) - \frac{1}{m\omega_p^2}(aP(t+\Delta t) + bP^3(t+\Delta t)) - \frac{\gamma_P}{m\omega_p}\dot{P}(t+\Delta t)\right]. \tag{57}$$

If we set $D = \frac{\hbar}{2em\omega_p}$, $E = \frac{\gamma_P}{m\omega_p}$ and $\eta = 1 + \frac{E\Delta t}{2}$ we can rewrite equation (57) as

$$\dot{P}(t+\Delta t) = \frac{1}{\eta}\left\{\dot{P}(t) + \frac{\ddot{P}(t)}{2}\Delta t + \frac{\Delta t}{2}\left[D\dot{\varphi}(t+\Delta t) - \frac{1}{m\omega_p^2}(aP(t+\Delta t) + bP^3(t+\Delta t))\right]\right\}. \tag{58}$$

We can do the same for equation (55). We then obtain :

$$\dot{\varphi}(t+\Delta t) = \dot{\varphi}(t) + \frac{\ddot{\varphi}(t)}{2}\Delta t + \frac{\Delta t}{2}\left[\frac{I}{I_c} - \sin(\varphi(t+\Delta t)) - \frac{1}{Q}\dot{\varphi}(t+\Delta t) - \frac{\omega_p}{I_c}\dot{P}(t+\Delta t)\right]. \tag{59}$$

Setting $A = \frac{1}{Q}$, $B = \frac{\omega_p}{I_c}$ and $\xi = 1 + \frac{A\Delta t}{2}$ we get :

$$\dot{\varphi}(t+\Delta t) = \frac{1}{\xi}\left\{\dot{\varphi}(t) + \frac{\ddot{\varphi}(t)}{2}\Delta t + \frac{\Delta t}{2}\left[\frac{I}{I_c} - \sin(\varphi(t+\Delta t)) - B\dot{P}(t+\Delta t)\right]\right\}. \tag{60}$$

We can then use equation (58) in equation (60). Setting $\zeta = \xi + \frac{BD\Delta t^2}{4\eta}$ we obtain:

$$\dot{\varphi}(t+\Delta t) = \frac{1}{\zeta}\Bigg\{\dot{\varphi}(t) + \frac{\ddot{\varphi}(t)}{2}\Delta t + \frac{\Delta t}{2}\left[\frac{I}{I_c} - \sin(\varphi(t+\Delta t))\right] + \frac{B\Delta t^2}{4\eta}\left[\frac{1}{m\omega_p^2}(aP(t+\Delta t) + bP^3(t+\Delta t)\right] - \frac{B\Delta t}{2\eta}\left[\dot{P}(t) + \frac{\ddot{P}(t)}{2}\Delta t\right]\Bigg\}. \tag{61}$$

We know all the values on the right side of equation (61), thus we can solve it. We can then use the value found for $\dot{\varphi}(t+\Delta t)$ in equation (58).

Thus we can then solve the system of equations (49) using the Velocity Verlet method by first calculating $\varphi(t+\Delta t)$ and $P(t+\Delta t)$ through equations (53) and (54). We can then use the results to calculate $\dot{\varphi}(t+\Delta t)$ via equation (61). Finally we can then calculate $P(t+\Delta t)$ using equation (58).

6. Results and Discussion

In this section we shall show our results and discuss them. For our simulation we have used the following values:

Table 1: The paramters used for the RCSJ simulation. These give an underdamped RCSJ.

$$\begin{aligned} R &= 0.5\mu\Omega \\ C &= 30mF \\ I_c &= 1A \\ t &= 1000s \\ \Delta t &= 0.01s \\ \omega_p &= 318.25MHz \\ P_s &= 0.1\frac{C}{m^2} \end{aligned}$$

The parameters in table 1 are chosen such that the RCSJ is underdamped.

6.1. The Standard RCSJ Model

If we run the simulation for a standard RCSJ with current $I = 1.5A$, initial conditions all equal to zero and the values in table 1 we get the following graphs for the phase and its first two derivatives.

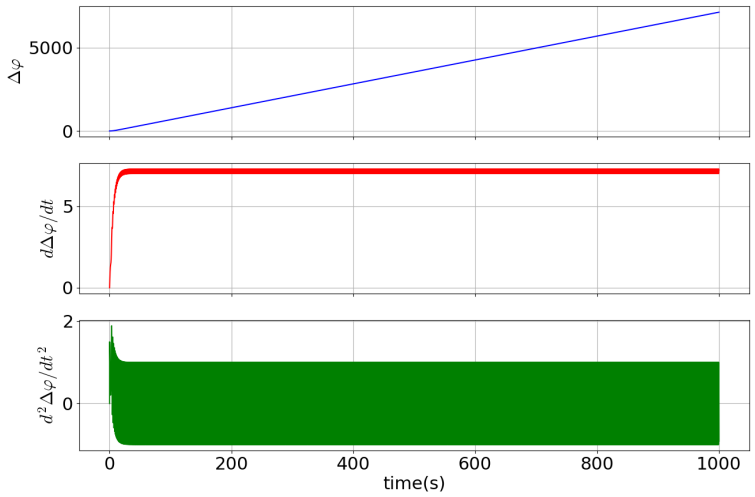

Figure 12: The phase difference $\Delta\varphi$, its first derivative $\frac{d\Delta\varphi}{dt}$ and its second derivative $\frac{d^2\Delta\varphi}{dt^2}$ are shown for a RCSJ with values given in table 1. The phase increases linearly. The first derivative makes small oscillations around a constant value. The second derivative oscillates around 0.

The phase increases linearly, its first derivative oscillates around a constant value and the second derivative oscillates around zero. This is what we expect intuitively as the first derivative of the phase is related to the voltage, and if we leave the junction long enough we expect a constant voltage. We can also understand these results by looking at the tilted washboard potential. When $I > I_c$ we can interpret the RCSJ as a particle moving downward along a potential given by (5) and illustrated in figure 5. We can see that the slope of the curve oscillates around a constant value. This is exactly what we see in figure 12. We simulate the RCSJ for a current sweep $0A \to 1.5A \to 0A \to -1.5A \to 0A$. We simulate with current steps of $0.01A$, $0.001A$ and $0.0001A$. At each current we take the average of the first two

derivatives of the phase and assume in the next step that the initial conditions are equal to these. For the initial condition of the phase we use its last value of the previous current. We take the absolute value of the voltage at negative current and plot this on the positive axis. We do this to see how good the two graphs overlap. For a normal RCSJ they should overlap perfectly.

For current step $0.01A$, the graph is given in figure 13. The graphs overlap perfectly and retrapping current $I_r = 0.27A$. In the simulation we observe that at the critical and retrapping currents the voltage does not jump instantaneously but has a few points in between.

For current step $0.001A$, the graph is given in figure 14. The graphs overlap perfectly and retrapping current $I_r = 0.387A$. The jumps at the critical current and the retrapping current are immediate.

For current step $0.0001A$ the graph is given in figure 15. The graphs do not overlap perfectly. There are two different retrapping currents $I_r = 0.3879$ and $I_r = 0.3805$. There is a gap of $0.0074A$. The jumps at the critical current and the retrapping current are also immediate.

For future simulations we shall use a current step of $0.001A$. The graph obtained for step size 0.001 is better than the graph of step size $0.01A$, while the run time for the simulation is significantly shorter than that of a simulation with step size $0.0001A$. We take note of the gap between the retrapping currents for step size $0.0001A$. This gives a rough idea of the error which occurs due to the numerical method.

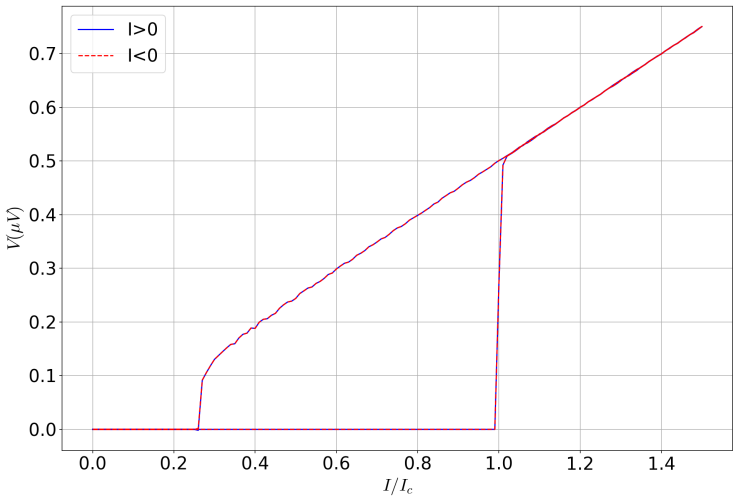

Figure 13: The Voltage current graph for a RCSJ with parameters from table 1. We have plotted the absolute value of the voltage against the absolute value of current. This makes it easy to see how well the graphs overlap. The solid blue line is the current voltage graph for positive current. The dotted red line is the current voltage graph for negative current. The current step size is $0.01A$. The graphs overlap perfectly. The retrapping current is $0.27A$. The jumps at the critical current and the retrapping current are not immediate, there are some intermediate points.

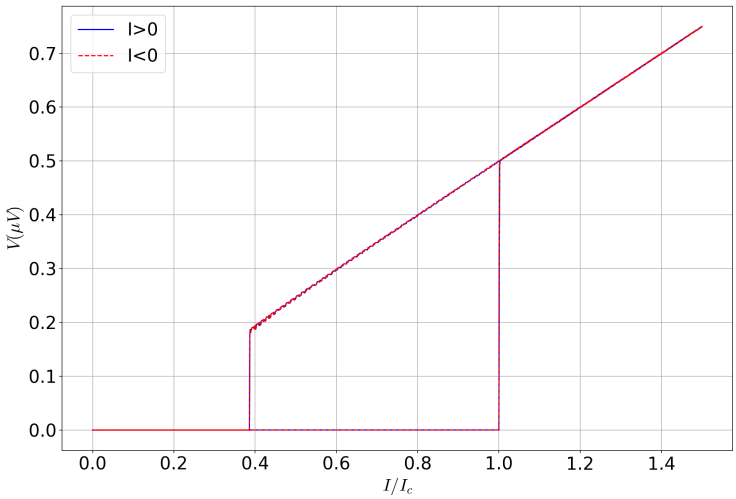

Figure 14: The Voltage current graph for a RCSJ with parameters from table 1. We have plotted the absolute value of the voltage against the absolute value of current. This makes it easy to see how well the graphs overlap. The solid blue line is the current voltage graph for positive current. The dotted red line is the current voltage graph for negative current. The current step size is $0.001A$. The graphs overlap perfectly. The retrapping current is $0.387A$. The jumps at the critical current and the retrapping current are immediate.

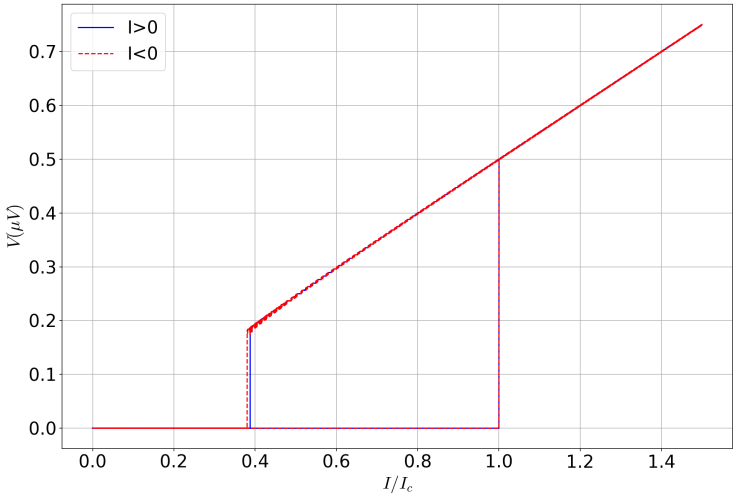

Figure 15: The Voltage current graph for a RCSJ with parameters from table 1. We have plotted the absolute value of the voltage against the absolute value of current. This makes it easy to see how well the graphs overlap. The solid blue line is the current voltage graph for positive current. The dotted red line is the current voltage graph for negative current. The current step size is $0.001A$. The graphs do not overlap perfectly. There are two different retrapping currents, $0.3879A$ and $0.3805A$. There is a gap 0.0074. The jumps at the critical current and the retrapping current are immediate.

6.2. The RCSJ Model with a Ferroelectric Barrier

We will now run the simulation for a RCSJ with a ferroelectric barrier. For this simulation we will assume that we have temperature $T = 90K$ and Curie temperature $T_c = 100K$. We shall first consider the case $\gamma_P = 0$. First we set $a_0 = 1$ and vary m. Then we set $m = 1$ and vary a_0. This is done for the current values $I = 0A$, $I = 0.5A$ and $I = 1.25A$. Since $I = 0A$ and $I = 0.5A$ are both smaller than the critical current we do not expect anything to happen for these currents.

Figure 16 shows the results for $m = 1$ and $a_0 = 1$. The frequency of the polarisation oscillation ω_{pol} is then of order $10\frac{rad}{s}$. Thus the two systems have frequencies differing by several magnitudes. The plasma frequency ω_p is much larger. This can be seen in the figure as in the duration of the simulation the voltage makes many oscillations but the polarisation and its first derivative do not change much.

Next we set $a_0 = 1$ and $m = 10^{-11}$, which gives ω_{pol} of order $10^6\frac{rad}{s}$. This is shown in figure 17. The two frequencies are now closer. P and $\frac{dP}{dt}$ both make more than two oscillations in the simulation time.

Now we set $a_0 = 1$ and $m = 10^{-17}$, which gives ω_{pol} of order $10^9\frac{rad}{s}$. The polarisation frequency is now larger than the plasma frequency. This is shown in figure 18. Now P and $\frac{dP}{dt}$ both make many oscillations in the simulation time.

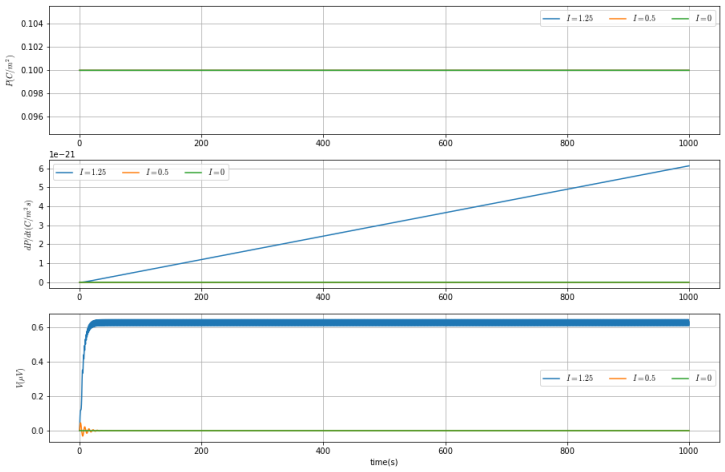

Figure 16: The polarisation P, its first derivative $\frac{dP}{dt}$ and the voltage V in a RCSJ junction are shown for a time span of $1000s$. We use the values in table 1 and $\gamma_p = 0$, $a_0 = 1$ and $m = 1$. We show the results for $I = 0A$, the green line, $I = 0.5A$, the orange line, and $I = 1.25A$ the blue line. We observe that for $I = 0A$ and $I = 0.5A$ not much happens. This is what we expect as both are smaller than the critical current. For $I = 1.25A$ we see that the voltage V oscillates around a certain value. $\frac{dP}{dt}$ seems to linearly increase however this increase is very small and P remains approximately constant. The difference of the plasma frequency ω_p and the polarisation frequency is clearly seen as V which oscillates at the plasma frequency makes many oscillations while $\frac{dP}{dt}$ and P do not even make one oscillation.

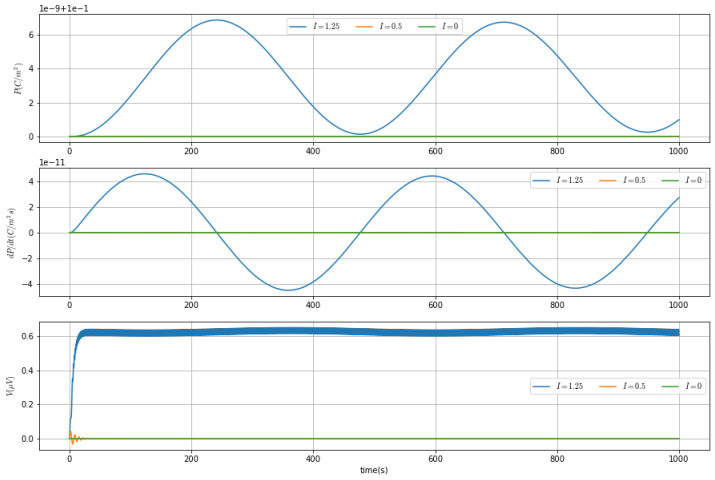

Figure 17: The polarisation P, its first derivative $\frac{dP}{dt}$ and the voltage V in a RCSJ junction are shown for a time span of $1000s$. We use the values in table 1 and $\gamma_p = 0$, $a_0 = 1$ and $m = 10^{-11}$. We show the results for $I = 0$A, the green line, $I = 0.5A$, the orange line, and $I = 1.25A$ the blue line. We observe that for $I = 0A$ and $I = 0.5A$ not much happens. This is what we expect as both are smaller than the critical current. For $I = 1.25A$ we see that the voltage V oscillates around a certain value. $\frac{dP}{dt}$ oscillates around zero. P oscillates around some value larger than P_s. The difference of the plasma frequency ω_p and the frequency of the polarisation oscillation is clearly seen as V which oscillates at the plasma frequency makes many oscillations while $\frac{dP}{dt}$ and P make approximately two oscillations.

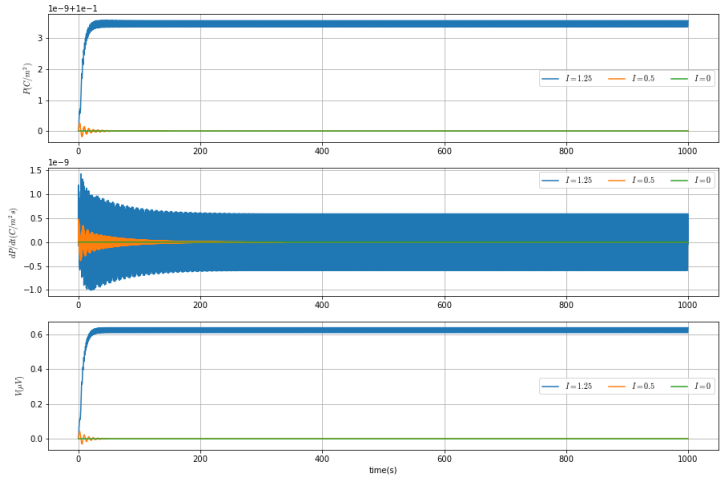

Figure 18: The polarisation P, its first derivative $\frac{dP}{dt}$ and the voltage V in a RCSJ junction are shown for a time span of $1000s$. We use the values in table 1 and $\gamma_p = 0$, $a_0 = 1$ and $m = 10^{-17}$. We show the results for $I = 0A$, the green line, $I = 0.5A$, the orange line, and $I = 1.25A$ the blue line. We observe that for $I = 0A$ and $I = 0.5A$ not much happens. This is what we expect as both are smaller than the critical current. For $I = 1.25A$ we see that the voltage V oscillates around a certain value, $\frac{dP}{dt}$ oscillates around zero, and P oscillates around some value larger than P_s. The plasma frequency ω_p and the frequency of the polarisation oscillation are now both very large as V, which oscillates at the plasma frequency, $\frac{dP}{dt}$ and P all make many oscillations.

We set $m = 1$ and look at $a_0 = 10^{11}$ and $a_0 = 10^{17}$. This is shown in figure 19 and figure 20 respectively. In both cases P and $\frac{dP}{dt}$ change very little. This is because we have made a_0 much larger while keeping V the same. As a result of this V has a negligible effect on the polarisation.

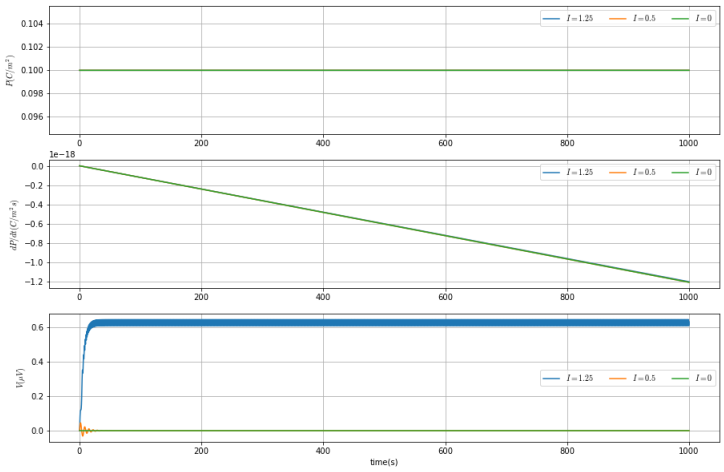

Figure 19: The polarisation P, its first derivative $\frac{dP}{dt}$ and the voltage V in a RCSJ junction are shown for a time span of $1000s$. We use the values in table 1 and $\gamma_p = 0$, $a_0 = 10^{11}$ and $m = 1$. We show the results for $I = 0A$, the green line, $I = 0.5A$, the orange line, and $I = 1.25A$ the blue line. For the polarisation and its first derivative we do not observe a difference between $I = 0A$, $I = 0.5A$ and $I = 1.25A$. This is because now a_0 is so large that the voltage V has a negligible effect on the polarisation. For $I = 1.25A$ we see that the voltage V oscillates around a certain value. $\frac{dP}{dt}$ seems to linearly decrease however this decrease is very small and P remains approximately constant. The difference of the plasma frequency ω_p and the frequency of the polarisation oscillation is clearly seen as V which oscillates at the plasma frequency makes many oscillations while $\frac{dP}{dt}$ does not even make one oscillation.

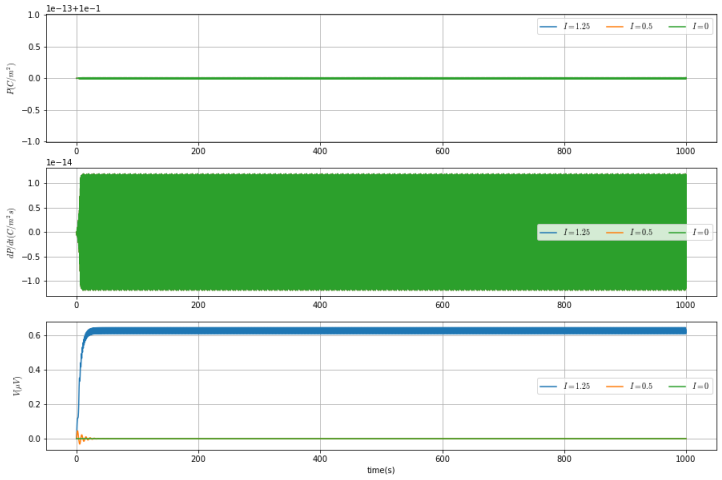

Figure 20: The polarisation P, its first derivative $\frac{dP}{dt}$ and the voltage V in a RCSJ junction are shown for a time span of $1000s$. We use the values in table 1 and $\gamma_p = 0$, $a_0 = 10^{17}$ and $m = 1$. We show the results for $I = 0A$, the green line, $I = 0.5A$, the orange line, and $I = 1.25A$ the blue line. For the polarisation and its first derivative we do not observe a difference between $I = 0A$, $I = 0.5A$ and $I = 1.25A$. This is because now a_0 is so large that the voltage V has a negligible effect on the polarisation. Both the frequency of the polarisation oscillation and the plasma frequency are large. $\frac{dP}{dt}$ and V both make many oscillations. However, $\frac{dP}{dt}$ is so small that P does not change significantly.

We want to run the above simulations for different currents. At each current we must pick a single value for all the variables which vary over time. For the phase we choose the last value, for its first two derivatives we take the time average. For the polarisation we take the time average, for its first derivative we take the maximum value and for its second derivative we take the last value. In figure 21 we show the $I - V$ curve for $a_0 = 1$ and different m and in figure 22 we to this for $m = 1$ and different a_0. We notice that varying m has a greater effect than varying a_0. This is because greatly increasing a_0 decreases the effect the voltage V has on the polarisation P.

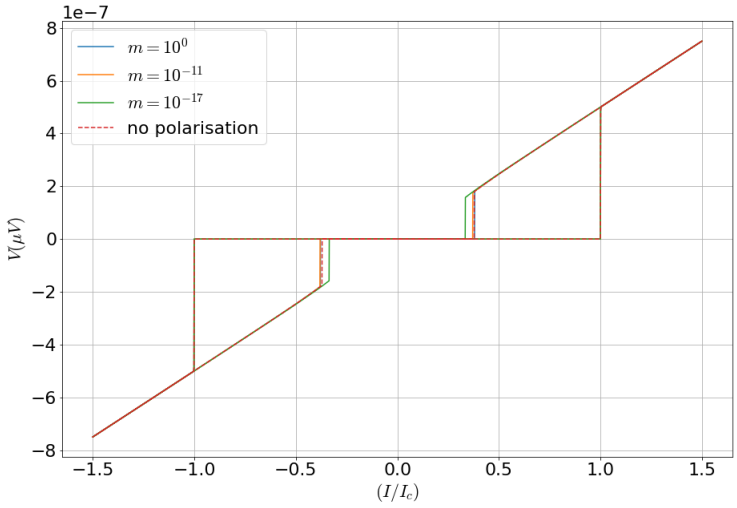

Figure 21: The voltage plotted against the current for an RCSJ with a ferroelectric. The parameters used are given in table 1. Furthermore we take $a_0 = 1$ and $\gamma_p = 0$. We show the graph for the normal RCSJ, the red dotted line and for the values $m = 1$, the blue line, $m = 10^{-11}$, the orange line and $m = 10^{-17}$, the green line. We see that $m = 10^{-17}$ has the greatest effect on the curve.

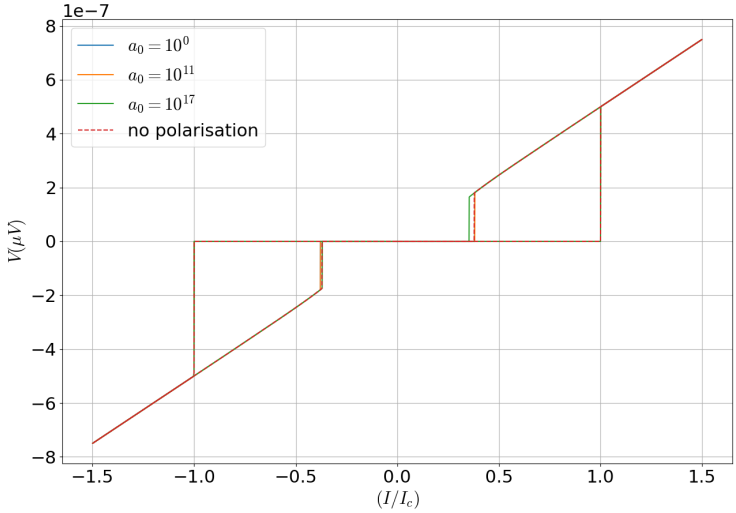

Figure 22: The voltage plotted against the current for an RCSJ with a ferroelectric. The parameters used are given in table 1. Furthermore we take $m = 1$ and $\gamma_p = 0$. We show the graph for the normal RCSJ, the red dotted line and for the values $a_0 = 1$, the blue line, $a_0 = 10^{11}$, the orange line and $a_0 = 10^{17}$, the green line. We see that effect on the curve is less than when changing m. This is due to the large a_0 making the effect of the voltage V negligible.

Last we apply a magnetic field perpendicular to the junction. We use parameters $a_0 = 1$, $m = 10^{-17}$ and $\gamma_p = 10^{-10}$. We look at the gap at the retrapping current for different magnetic flux values. The result is shown in figure 23. Int his graph we do not directly see a pattern. We also notice that the current gap values are close to the

current gap for a normal RCSJ at current step $0.0001A$ which is $0.0074A$. It is therefore possible that these gaps are due to the numerical method used.

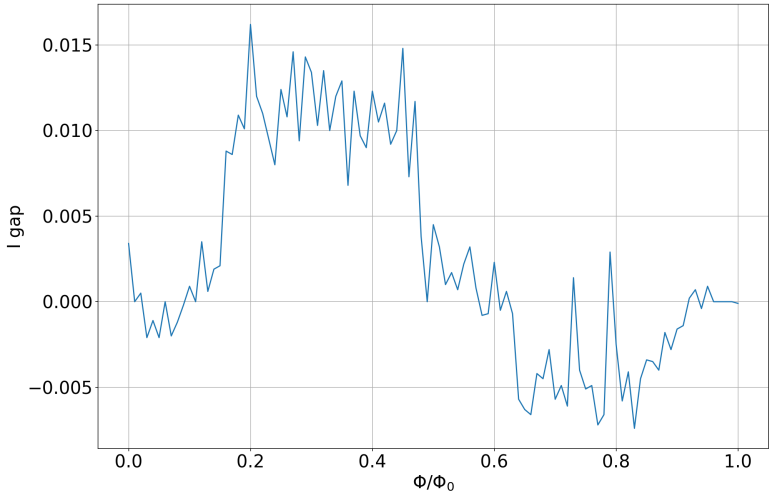

Figure 23: The gap in the retrapping current is plotted against the flux Φ divided by the magnetic flux quantum Φ_0. Here we have used current step $0.001A$, $a_0 = 1$, $m = 10^{-17}$ and $\gamma_p = 10^{-10}$. The gaps here are close to the gap for a normal RCSJ with current step $0.0001A$ which is $0.0074A$. There is also no directly obvious pattern.

7. Conclusion

We have run simulations using the Velocity Verlet method on a normal RCSJ with parameters $R = 0.5\mu\Omega$, $C = 30mF$ and $I_c = 1A$. These parameters give a underdamped RCSJ. We use time steps $\Delta t = 0.01s$ and a time span $t = 1000s$. First we run the simulation for a single current value larger than the critical current, $I = 1.5A$. This simulation gives results which coincide with the theory. Next we perform a current sweep where we go from $0A$ to $1.5A$, then from $1.5A$ to $0A$, then from $0A$ to $-1.5A$, and finally from $-1.5A$ to $0A$. We plot the voltage against the current. This is done for current steps of sizes $0.01A$, $0.001A$, and $0.0001A$. For $0.01A$ the jumps in the graph are not instantaneous and have a few intermediate points, causing a line with a slope rather than a vertical line. We would like a graph with a immediate jump and thus a vertical line. This occurs for both the simulation with step size $0.001A$ and the simulation with step size $0.0001A$. We choose to use step size $0.001A$ in future simulations as simulations using step size $0.0001A$ consume a significant amount of time. For step size $0.0001A$ we notice that there is a gap of $0.0074A$ between the the positive and negative retrapping currents. Theoretically there should be no gap so our results contradict the theory. We conclude that this gap is due to the numerical method.

Next we simulate a RCSJ with a ferroelectric barrier. We observe that when $a_0 = 1$, the polarisation is only affected for small m. This can be explained through the frequency of the polarisation oscillation. If this frequency is too small, the polarisation will not change significantly in the time span of the simulation. The time span for the simulation is chosen based on the plasma frequency, thus the frequency of the oscillation of the polarisation must be close tot the plasma frequency for the polarisation to change significantly and have an effect on the RCSJ. We also notice that varying either m or a_0 and keeping the other constant leads to different results even when the polarisation frequency is the same. Specifically, varying m has a greater effect on the current voltage graph than varying a_0. This is because m, which can be interpreted as the inertia of the polarisation, does not have an effect on the free energy of the polarisation while a_0 does. Therefore if a_0 is too large, the voltage will be negligible in the polarisation free energy and the polarisation will be unaffected by the junction. From the graphs we observe a difference between a normal RCSJ and a RCSJ with a ferroelectric barrier at the retrapping current.

Lastly the effect of a magnetic field on a RCSJ with a ferroelectric barrier is studied. We look at the gap between the positive and negative retrapping current. Such a gap would mean that there is some region in which we have superconductivity in one direction but not the other, which is the superconducting diode effect. We do not observe a recognizable pattern. The gaps are also close to the gap observed for a normal RCSJ. Thus the numerical method used is not accurate enough to detect a gap between retrapping currents because if these gaps are present, they are to small to distinguish from the numerical error.

To enhance the accuracy of our simulations, we propose experimenting with alternative numerical methods, such as the fourth-order Runge-Kutta method or Euler's method, to assess whether they can effectively reduce the numerical error. Additionally, another approach would be to decrease the time and current step sizes to determine if this leads to improved accuracy in our results. However, there is a trade-off as this adjustment could demand more computing power, potentially prolonging simulation times. An alternative approach would be to improve the code used for the simulation.

In future research, interesting idea would be to consider a magnetic field which both depends on and affects the polarisation. Taking this idea one step further, an intriguing direction to pursue is the analysis of a Josephson junction with a multiferroic barrier. Multiferroic materials possess both ferroelectric and ferromagnetic characteristics. Hence, they can possess both an electric and magnetic dipole moment, making them an excellent choice for further study.

References

[1] Alex I Braginski. Superconductor electronics: Status and outlook. *Journal of superconductivity and novel magnetism*, 32:23–44, 2019.

[2] Göran Wendin. Quantum information processing with superconducting circuits: a review. *Reports on Progress in Physics*, 80(10):106001, 2017.

[3] Yuta Miyasaka, Ryo Kawarazaki, Hideki Narita, Fuyuki Ando, Yuhei Ikeda, Ryusuke Hisatomi, Akito Daido, Yoichi Shiota, Takahiro Moriyama, Youichi Yanase, et al. Observation of nonreciprocal superconducting critical field. *Applied Physics Express*, 14(7):073003, 2021.

[4] Hideki Narita, Jun Ishizuka, Ryo Kawarazaki, Daisuke Kan, Yoichi Shiota, Takahiro Moriyama, Yuichi Shimakawa, Alexey V Ognev, Alexander S Samardak, Youichi Yanase, et al. Field-free superconducting diode effect in noncentrosymmetric superconductor/ferromagnet multilayers. *Nature Nanotechnology*, 17(8):823–828, 2022.

[5] Christian Baumgartner, Lorenz Fuchs, Andreas Costa, Simon Reinhardt, Sergei Gronin, Geoffrey C Gardner, Tyler Lindemann, Michael J Manfra, Paulo E Faria Junior, Denis Kochan, et al. Supercurrent rectification and magnetochiral effects in symmetric josephson junctions. *Nature nanotechnology*, 17(1):39–44, 2022.

[6] Heng Wu, Yaojia Wang, Yuanfeng Xu, Pranava K Sivakumar, Chris Pasco, Ulderico Filippozzi, Stuart SP Parkin, Yu-Jia Zeng, Tyrel McQueen, and Mazhar N Ali. The field-free josephson diode in a van der waals heterostructure. *Nature*, 604(7907):653–656, 2022.

[7] Miraceti. English: Single Josephson junction; A,B superconductors; C thin insulator Čeština: Josephsonův přechod; A,B supravodiče; C tenká vrstva izolantu, February 2011.

[8] Bjorn Ahlgren and David Byström. A study of josephson junction characteristics. 2011.

[9] Michael Tinkham. *Introduction to Superconductivity*. Dover Publications, 2 edition, June 2004.

[10] Francesco Tafuri. *Fundamentals and frontiers of the Josephson effect*, volume 286. Springer Nature, 2019.

[11] John R. Kirtley. *Magnetic Field Effects in Josephson Junctions*, pages 209–233. Springer International Publishing, Cham, 2019.

[12] Jakob Blomgren. The josephson effect. 2005.

[13] Jon F Ihlefeld. Fundamentals of ferroelectric and piezoelectric properties. In *Ferroelectricity in Doped Hafnium Oxide: Materials, Properties and Devices*, pages 1–24. Elsevier, 2019.

[14] P. Chandra and P. B. Littlewood. A landau primer for ferroelectrics, 2006.

[15] Stephen J Blundell and Katherine M Blundell. *Concepts in thermal physics*. Oup Oxford, 2010.

[16] SA Ktitorov, VA Trepakov, AF Ioffe, L Jastrabik, and L Soukup. Josephson effect in a junction with a ferroelectric. *Ferroelectrics*, 157(1):387–392, 1994.

[17] Ilhom Rahmonov, Nikolay Chtchelkatchev, and Yury Sjukrinov. The resonance properties of superconductor/-ferroelectric/superconductor heterostructure. In *7th International Workshop on Numerical Modelling of High Temperature Superconductors*, 2021.

A. Parameters

Quantity	Meaning	Symbol	Unit
Current	The amount of charge flowwing per second	I	Ampere$A$
Voltage	The electric potential	V	Volt (V)
Resistance	The electrical resistance	R	Ohm (Ω)
Capacitance	The electrical capacitance	C	Farad(F)
Polarisation	The electrical Polarisation	P	Coulomb/squaremeter (C/m^2)
time	the elapsed time	t	second (s)
Phase	The phase difference between two wavefunctions	φ	radians(rad)
Angular frequency	The rate at which teh phase changes	ω	radians/second (rad/s)
frequency	The number of oscillations per second	f	Hertz (Hz) or ($1/s$)
Magnetic Field	The magnetic field	B	Tesla (T)
Magnetic Flux	The normal component of a magnetic field integrated over an area	Φ	Tesla squaremeter (Tm^2)
Magnetic vector potential	The vector potential of the magnetic field defined by $\mathbf{B} = \nabla \times \mathbf{A}$	A	Tesla meter (Tm)

B. Python Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun  8 17:05:34 2023

@author: Keane
"""

# In[Import packages]

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
from scipy.integrate import odeint
from scipy import constants
from numba import jit
from numba import njit
import time

# In[Physical constants]

h       = constants.h                                          # h                 = 6.62607015e-34
hbar    = constants.hbar                                       # hbar              = 1.0545718176461565e-34
pi      = constants.pi                                         # Pi                = 3.141592653589793
e       = constants.e                                          # Elementary charge = 1.602176634e-19
Phi_0   = constants.physical_constants['mag. flux quantum'][0]  # Flux quantum      = 2.067833848e-15
cVphi   = hbar/(2*e)                                           # V/dphidt          = 3.2910597847545335e-16

# In[Independent parameters]

C           = 3*10**(-2)     # Capacitance
R           = 5*10**(-7)     # Resistance

gamma_p     = 1*10**(-5)     # Damping factor of the polarisation

Tcurie = 100
T      = 10
P_0    = 1e-5
a_0    = 1e13

# In[Dependent parameters]

@njit
def beta_c (I_c, R, C) :
    return I_c*C*R**2/cVphi

@njit
def omega_p (I_c, C) :
    """Frequency of the phase omegap = (2eI_c/Chbar)^1/2"""
    return np.sqrt(I_c/(cVphi*C))

@njit
def Q (I_c, R, C) :
    """Damping of the phase Q = omegap*R*C"""
    return np.sqrt(I_c*C*R**2/cVphi)

@njit
```

```
def m_eff (I_c, C) :
    """Inertia of the polarisation m_eff = 1/omegap"""
    return np.sqrt(cVphi*C/I_c)

@njit
def A_p (a0, T) :
    """First Landau parameter A_p = a0(t-T_curie)"""
    return a0*(T - T_Curie)  # T<T_Curie,a0 ~ 10^13

@njit
def B_p (A, P0) :
    """Second Landau parameter B_p = -A/P0^2"""
    return -A/P0**2


# In[Data points]

@njit
def current_points (I_max, num_steps) :
    """" Makes num_steps points for intervals (0, I_max), (I_max, 0),
    (0, -I_max), (-I_max, 0) and returns them in that order.
    Also returns the sweep concatenated in the same order and reverse order.
    Return object is a tuple"""

    I_forward      = np.linspace(0, I_max, num = num_steps)                                  # Bias
     current 0 - 1.5*I_c
    I_backward     = np.flip(I_forward)                                                      # Bias
     current 1.5*I_c - 0
    I_forwardneg   = np.linspace(0, -I_max, num = num_steps)                                 # Bias
     current 0 - -1.5*I_c
    I_backwardneg  = np.flip(I_forwardneg)                                                   # Bias
     current -1.5*I_c - 0
    I_cycle        = np.concatenate((I_forward,I_backward,
                                     I_forwardneg,I_backwardneg)) # Bias current cycle 0 -> 1.5*
    I_C ->
                                                                  #                    0 -> -1.5*
    I_c -> 0
    I_backw_cycle  = np.concatenate((I_forwardneg, I_backwardneg,
                                     I_forward, I_backward))

    I_pos_cycle  = np.concatenate((I_forward, I_backward))     # Bias current cycle 0 -> -1.5*I_C
     ->
                                                                  #                    0 -> 1.5*
    I_c -> 0

    I_neg_cycle  = np.concatenate((I_forwardneg, I_backwardneg))

    return (I_forward, I_backward, I_forwardneg,
            I_backwardneg, I_cycle, I_backw_cycle, I_neg_cycle, I_pos_cycle)

@njit
def time_points (tspan, stepsize) :
    """ Makes steps of size stepsize from 0 to tspan.Calculates the staring
    point for averaging. Returns an array with the time points and

    the starting point for overaging in a tuple"""


    num_steps = np.array([(tspan/stepsize)]).astype(np.int_)

    t         = np.linspace(0, tspan, num = num_steps[0]+1)

    return t

@njit
```

```
def I_gap (V, I) :
    ip = np.nonzero(V > 1e-8)
    i  = np.nonzero(V < -1e-8)
    I_c_diffh = (np.absolute(I[ip[0][0]]) - np.absolute(I[i[0][0]]))
    I_c_diffl = (np.absolute(I[ip[0][-1]]) - np.absolute(I[i[0][-1]]))

    return (I_c_diffh, I_c_diffl)

# In[Numerical Methods]

@njit
def Velocity_Verlet_RCSJ (R, C, I_c, I, time_step, tspan, initc) :

    t = time_points(tspan, time_step)
    damp = Q(I_c, R, C)
    dt = time_step
    B = 1/(1 + dt/(2*damp))

    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi = np.zeros((3, num_steps))
    # Assign initial conditions
    phi[0][0] = initc[0]
    phi[1][0] = initc[1]
    phi[2][0] = initc[2]

    # Use the velocity verlet algorithm
    for i in range(num_steps-1) :
        phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
        phi[1][i+1] = B*(phi[1][i] + phi[2][i]*dt/2 + (I/I_c - np.sin(phi[0][i+1]))*dt/2)
        phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp

    return (t, phi)
# In[]
ts = time.perf_counter_ns()
data = Velocity_Verlet_RCSJ(R, C, 1, 1.5, 0.1, 100, np.array([0,0,0]))
tf = time.perf_counter_ns()
tr = tf-ts
print(tr*1e-9)

# In[Plot]



t = data[0]
x = data[1][0]
v = data[1][1]
a = data[1][2]

%matplotlib qt5

rcParams.update({'font.size': 22})
rcParams["mathtext.fontset"] = "cm"

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, sharex= True)



ax1.plot(t,x, 'b')
#ax1.set_xlabel('time(s)')
ax1.set_ylabel(r'$\varphi$')
ax1.grid()

ax2.plot(t, v, 'r')
```

```
#ax2.set_xlabel('time(s)')
ax2.set_ylabel(r'$d\varphi/dt$')
ax2.grid()
#ax2.set_yticks([0,  2,  4, 6, 7, 8])


ax3.plot(t, a, 'g')
ax3.set_xlabel('time(s)')
ax3.set_ylabel(r'$d\varphi^2/dt^2$')
ax3.grid()
#ax3.set_yticks([-1, -2, 0, 1, 2])

fig.set_size_inches(15,10)

plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/RCSJ phase behaviour.png')

# In[ Velocity verlet RCSJ system]
@njit
def VV_RCSJ (R, C, I_c, I_max, num_current_steps, time_step, tspan, initc, avstart) :

    t = time_points(tspan, time_step)
    dt = time_step

    # Calculate number of current points as an integer
    I_data  = current_points(I_max, num_current_steps)
    I_steps = current_points(I_max, num_current_steps)[4].shape[0]
    CPR = np.zeros(I_steps)
    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi = np.zeros((3, num_steps))
    # Assign initial conditions
    phi[0][0] = initc[0]
    phi[1][0] = initc[1]
    phi[2][0] = initc[2]

    if I_c == 0 :
        labda = 1 + (dt/(2*R*C))
        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            mu = I/(cVphi*C)
            for i in range(num_steps-1) :
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                phi[1][i+1] = (1/labda)*(phi[1][i] + phi[2][i]*dt/2 + mu*dt/2)
                phi[2][i+1] = mu - phi[1][i+1]/(R*C)

            avs = np.array([avstart*tspan]).astype(np.int_)[0]
            phi_avg   = np.average(phi[0][avs:])
            dphi_avg  = np.average(phi[1][avs:])
            d2phi_avg = np.average(phi[2][avs:])
            V         = dphi_avg*cVphi
            CPR[j] = V
            phi[0][0],phi[1][0] = phi[0][-1], phi[1][-1]

    else :
        damp = Q(I_c, R, C)
        B = 1/(1 + dt/(2*damp))
        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            for i in range(num_steps-1) :
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                phi[1][i+1] = B*(phi[1][i] + phi[2][i]*dt/2 + (I/I_c - np.sin(phi[0][i+1]))*dt/2)
                phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp
```

```
244
245             avs = np.array([avstart*tspan]).astype(np.int_)[0]
246             phi_avg   = np.average(phi[0][avs:])
247             dphi_avg  = np.average(phi[1][avs:])
248             dphi_max  = np.max(phi[1])
249             d2phi_avg = np.average(phi[2][avs:])
250             V         = dphi_avg*cVphi*omega_p(I_c, C)
251             CPR[j] = V
252             phi[0][0],phi[1][0] = phi[0][-1], dphi_avg
253
254     return (I_data, CPR)
255 # In[Retrieve data]
256 ncp = 15001
257 ts = time.perf_counter_ns()
258 ic = np.array([0, 0, 0])
259 RCSJ = VV_RCSJ(R, C, 1, 1.5, ncp, 0.01, 100, ic, 0.5)
260 tf = time.perf_counter_ns()
261 tr = (tf - ts)*1e-9
262 print(tr)
263 x = RCSJ[0][4]
264 y = RCSJ[1]
265
266 ipos = np.nonzero(y > 1e-8)
267 ineg  = np.nonzero(y < -1e-8)
268 Iretrappos = x[ipos[0][-1]]
269 Iretrapneg = x[ineg[0][-1]]
270
271 print(Iretrappos, Iretrapneg)
272
273 g = I_gap(y, x)
274 print(g)
275
276
277 # In[Plot data]
278
279 font = {'family': 'serif',
280         'color':  'black',
281         'weight': 'normal',
282         'size': 20,
283         }
284
285 %matplotlib qt5
286 plt.figure()
287 plt.plot(x, y, 'k')
288 plt.xlabel('$I / I_c$', fontdict = font)
289 plt.ylabel('V (V)', fontdict = font)
290 plt.grid()
291 plt.show()
292 # Till here everything works fine
293
294 # In[Plot data]
295 x1 = x[:ncp]
296 x2 = x[ncp:2*ncp]
297 x3 = x[2*ncp:3*ncp]
298 x4 = x[3*ncp:]
299 y1 = y[:ncp]*1e6
300 y2 = y[ncp:2*ncp]*1e6
301 y3 = y[2*ncp:3*ncp]*1e6
302 y4 = y[3*ncp:]*1e6
303
304 rcParams.update({'font.size': 22})
305
306 plt.figure(figsize=[15, 10], dpi = 150)
307 plt.plot(x1, y1, '-', label = '0 -> 1.5')
308 plt.plot(x2, y2, '-', label = '1.5 -> 0')
```

```
plt.plot(x3, y3, '-', label = '0 -> -1.5')
plt.plot(x4, y4, '-', label = '-1.5 -> 0')
plt.xlabel('$I/I_c$')
plt.ylabel('$V (\mu V)$')
plt.legend(loc = 'best')
plt.grid()
#plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/RCSJ different colors 2.png')
print(Iretrappos, Iretrapneg)
# In[RCSJ with flux]
@njit
def RCSJ_f(flux, R, C, I_max, I_0,  num_current_steps, time_step, tspan, initc, avstart ) :

    fsteps = np.array([flux.shape[0]]).astype(np.int_)[0]
    I_data = current_points(I_max, num_current_steps)
    I_stepsf = I_data[4].shape[0]
    VIf = np.zeros((fsteps,I_stepsf))

    for i in range(fsteps) :
        I_c = I_0*np.abs(np.round(np.sinc(flux[i]), 16))
        data = VV_RCSJ(R, C, I_c, I_max, num_current_steps, time_step, tspan, initc, avstart)
        VIf[i] = data[1]
    return (I_data, VIf)

# In[ Retrieve flux data]
ts = time.perf_counter_ns()
fl = np.linspace(0, 1, num = 6)
initc = np.array([0, 0, 0])
fl_data = RCSJ_f(fl, R, C, 1.5, 1, 1001, 0.01, 100, initc, 0.5)
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)
# In[]
flux = np.linspace(0, 2, num = 9)
for i in range(len(flux)) :
    I_c = np.abs(np.round(np.sinc(flux[i]), 16))
    print(I_c, i*0.2)
# In[]
flvals = np.linspace(-5, 5, num = 10001)
I_vals =np.abs( np.sinc(flvals) )

%matplotlib qt5

rcParams.update({'font.size': 35})

plt.figure(figsize=[15, 10], dpi = 150)
plt.plot(flvals, I_vals)
plt.xlabel('$\Phi/\Phi_0$')
plt.ylabel('$I_c/I_{c0}$')
plt.xticks([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/Fraunhofer pattern.png')
# In[]
%matplotlib qt5
x = fl_data[0][4]
yd = fl_data[1]*1e6

rcParams.update({'font.size': 22})

plt.figure(figsize=[15, 10], dpi = 150)
k = 0
for p in yd :
    l = np.round(fl[k],3)
    plt.plot(x, p, label = l )
```

```
    k = k+1

plt.xlabel('$I/I_c$')
plt.ylabel('$V (\mu V)$')
plt.legend(loc = 'upper left', ncol=3)
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/RCSJ with flux.png')

# In[]

Tcurie = 100
T      = 10
P_0    = 0.1
a_0    = 1e-5
A      = a_0*(T-Tcurie)
B      = ((-A)/(P_0**2))
P      = np.linspace (-0.25, 0.25, num=2001)
F00 =  B/4*P**4
F0  = A/2*P**2 + B/4*P**4

rcParams.update({'font.size': 22})

plt.figure(figsize=[15, 10], dpi = 150)
plt.plot(P, F00, 'g', label = '$T=T_c$')
plt.plot(P, F0, 'b', label = '$T<T_c$')
plt.xlabel('$P (C/m^2)$')
plt.ylabel('$F  (J)$')
plt.legend(loc = 'best')
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/Polarisation free energy for ferroelectric and
    paraelectric.png')


# In[Velocity verlet polarisation]
Tcurie = 100
T      = 10
P_0    = 0.1
a_0    = 1e-5
A      = a_0*(T-Tcurie)
B      = ((-A)/(P_0**2))
P      = np.linspace (-0.25, 0.25, num=2001)
Vm = 2*np.abs(A)*P_0/(3*np.sqrt(3))
F0 = A/2*P**2 + B/4*P**4
F = A/2*P**2 + B/4*P**4 -Vm*P
F2 = A/2*P**2 + B/4*P**4 +Vm*P
# In[]
%matplotlib qt5

rcParams.update({'font.size': 22})

plt.figure(figsize=[15, 10], dpi = 150)
plt.plot(P, F0, 'g', label = '0 Voltage')
plt.plot(P, F, 'r-', label= 'Positive Critical Voltage')
plt.plot(P, F2, 'b', label= 'NegativeCritical Voltage')

plt.xlabel('$P (C/m^2)$')
plt.ylabel('$F  (J)$')
plt.legend(loc = 'best')
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/Polarisation free energy with 0 and critical
    voltage.png')
```

```
print(Vm)

# In[]
%matplotlib qt5
V = np.linspace(0, 1e-4, num = 6)

rcParams.update({'font.size': 22})

plt.figure(figsize=[15, 10], dpi = 150)

for i in V :
    f = np.round(i, 7)
    F = A/2*P**2 + B/4*P**4 - f*P
    plt.plot(P, F, label = f)


plt.xlabel('$P (C/m^2)$')
plt.ylabel('$F (J)$')
#plt.legend(loc = 'best')
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/Polarisation free energy with different voltages.
    png')


# In[]
@njit
def pol (m, gamma, a0, p0, V, dt, tspan, ic) :
    tst = np.array([tspan/dt + 1]).astype(np.int_)[0]
    t   = time_points(tspan, dt)
    omp = omega_p(1, C)
    A   = gamma/(m*omp)
    a   = a0*(T-Tcurie)
    b   = ((-a)/(p0**2))
    p = np.zeros((3,tst ))
    p[0][0] = ic[0]
    p[1][0] = ic[1]
    p[2][0] =  (1/(m*omp**2))*(V - a*(p[0][0]) - b*(p[0][0])**3) - A*p[1][0]
    for i in range(tst-1) :
        p[0][i+1] = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2
        p[1][i+1] = (1/(1 + A*dt/2))*( p[1][i] + p[2][i]*dt/2 + V*dt/(2*m*omp**2) + (dt/(2*m*omp
    **2))*(-a*(p[0][i+1]) - b*(p[0][i+1])**3) )
        p[2][i+1] = (1/(m*omp**2))*(V - a*(p[0][i+1]) - b*(p[0][i+1])**3) - A*p[1][i+1]

    return (t, p)

# In[]
ts = time.perf_counter_ns()
ic = np.array([1e-8, 0, 0])
po = pol(1e-11, 0, 1, 1, 1e-8, 0.01, 1000, ic)
t = po[0]
x = po[1][0]
v = po[1][1]
a = po[1][2]
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)


# In[]
%matplotlib inline
plt.figure()
plt.plot(t, x)
plt.xlabel('t')
plt.ylabel('P')
```

```
plt.grid()
plt.show()

plt.figure()
plt.plot(t, v)
plt.xlabel('t')
plt.ylabel('dP/dt')
plt.grid()
plt.show()

plt.figure()
plt.plot(t, a)
plt.xlabel('t')
plt.ylabel('d2P/dt2')
plt.grid()
plt.show()

# In[]
tspan = 10
dt = 0.01
gamma = 0
m = 1
a0 = 1
T = 10
Tcurie = 100
V = 0.1
p0 = 1

ts = time.perf_counter_ns()
ic = np.array([1, 0, 0])

tst = np.array([(tspan/dt)+1]).astype(np.int_)[0]
t = time_points(tspan, dt)
A = gamma/m
a      = a0*(T-Tcurie)
b      = ((-a)/(p0**2))
p = np.zeros((3,tst ))
p[0][0] = ic[0]
p[1][0] = ic[1]
p[2][0] = ic[2]
for i in range(tst-1) :
    p[0][i+1] = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2
    p[1][i+1] = (1/(1 + A))*( p[1][i] + p[2][i]*dt/2 + V*dt/2 + (dt/(2*m))*(-a*(p[0][i+1]) - b*(p
    [0][i+1])**3) )
    p[2][i+1] = (1/m)*(V - a*(p[0][i+1]) - b*(p[0][i+1])**3) - A*p[1][i+1]

tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)
x = p[0]
v = p[1]
a = p[2]

# In[]
%matplotlib inline
plt.figure()
plt.plot(t, x)
plt.xlabel('t')
plt.ylabel('P')
plt.grid()
plt.show()

plt.figure()
plt.plot(t, v)
plt.xlabel('t')
```

```
plt.ylabel('dP/dt')
plt.grid()
plt.show()

plt.figure()
plt.plot(t, a)
plt.xlabel('t')
plt.ylabel('d2P/dt2')
plt.grid()
plt.show()



# In[]
@njit
def VV_RCSJ_pt (R, C, gamma, m, a0, p0, I_c, I, time_step, tspan, initc) :
    # Equation constants
    t = time_points(tspan, time_step)
    damp = Q(I_c, R, C)
    omegap = omega_p(I_c, C)
    dt = time_step
    a      = a0*(T-Tcurie)
    b      = ((-a)/(p0**2))

    # Velocity verlet constants
    A   = -1/damp
    B   = -omegap/I_c
    D   = cVphi/(omegap*m)
    E   = - gamma/(m*omegap)
    eta = 1 - E*dt/2
    xi  = 1 - A*dt/2 - B*D*dt**2/(4*eta)

    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi, p = np.zeros((3, num_steps)), np.zeros((3, num_steps))
    # Assign initial conditions
    phi[0][0] = initc[0]
    phi[1][0] = initc[1]
    phi[2][0] = I/I_c - np.sin(phi[0][0]) - phi[1][0]/damp - (omegap/I_c)*p[1][0]

    p[0][0] = initc[3]
    p[1][0] = initc[4]
    p[2][0] = (cVphi/(m*omegap))*phi[1][0] - (gamma/(m*omegap))*p[1][0] - (1/(m*omegap**2))*(a*p
    [0][0] + b*p[0][0]**3)

    for i in range(num_steps-1) :
        phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
        p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2

        phi[1][i+1] = (1/xi)*( phi[1][i] + phi[2][i]*dt/2 + (B*dt/(2*eta))*(p[1][i] + p[2][i]*dt
    /2) + (I/I_c - np.sin(phi[0][i+1]))*dt/2 + (B*dt**2/(4*eta))*(1/(m*omegap**2))*(-a*p[0][i+1]
    - b*p[0][i+1]**3) )
        p[1][i+1]   = (1/eta)*( p[1][i] + p[2][i]*dt/2 + (D*dt/2)*phi[1][i+1] + (1/(m*omegap**2))
    *(-a*p[0][i+1] - b*p[0][i+1]**3) )

        phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp - (omegap/I_c)*p[1][i+1]
        p[2][i+1]   = (cVphi/(m*omegap))*phi[1][i+1] - (gamma/(m*omegap))*p[1][i+1] - (1/(m*
    omegap**2))*(a*p[0][i+1] + b*p[0][i+1]**3)

    return (t, p, phi)

# In[RCSJ polarisation for 1 current]
ts = time.perf_counter_ns()
Pin = 10
```

```
ic = np.array([0, 0, 0, Pin, 0, 0])
RCSJpt = VV_RCSJ_pt(R, C, 0, 1e-11, 1, Pin, 1, 1.5, 0.1, 1000, ic)
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)
t = RCSJpt[0]
xphi = RCSJpt[2][0]
vphi = cVphi*omega_p(1, C)*RCSJpt[2][1]
aphi = RCSJpt[2][2]
xp = RCSJpt[1][0]
vp = RCSJpt[1][1]
ap = RCSJpt[1][2]

# In[RCSJ polarisation for different currents]
%matplotlib inline
ts = time.perf_counter_ns()
k = np.linspace(0, 1.5, num=16)
plt.figure()
for i in range(len(k)) :
    curr = k[i]

    Pin = 10
    ic = np.array([0, 0, 0, Pin, 0, 0])
    RCSJpt = VV_RCSJ_pt(R, C, 1e-5, 1e-11, 1, Pin, 1, curr, 0.1, 1000, ic)

    t = RCSJpt[0]
    xphi = RCSJpt[2][0]
    vphi = cVphi*omega_p(1, C)*RCSJpt[2][1]
    aphi = RCSJpt[2][2]
    xp = RCSJpt[1][0]
    vp = RCSJpt[1][1]
    ap = RCSJpt[1][2]

    val = np.round(k[i], 3)
    plt.plot(t, vp, label = val)

tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)

plt.xlabel('t')
plt.ylabel('dP/dt')
plt.legend(loc = 'best')
plt.grid()
plt.show()

# In[]

%matplotlib qt5

ts = time.perf_counter_ns()

plt.figure()

Pin = 1e-8
ic = np.array([0, 0, 0, Pin, 0, 0])
RCSJpt = VV_RCSJ_pt(R, C, 0, 1e-11, 1, Pin, 1, 1.5, 1e-1, 1000, ic)

t = RCSJpt[0]
xphi = RCSJpt[2][0]
vphi = cVphi*omega_p(1, C)*RCSJpt[2][1]
aphi = RCSJpt[2][2]
xp = RCSJpt[1][0]
vp = RCSJpt[1][1]
ap = RCSJpt[1][2]
```

```
689
690 plt.plot(t, vp, '-', label = tstep)
691
692 plt.xlabel('t')
693 plt.ylabel('dP/dt')
694 plt.legend(loc = 'best')
695 plt.grid()
696 plt.show()
697
698 tf = time.perf_counter_ns()
699 tr = (tf - ts)*1e-9
700 print(tr)
701
702 # In[]
703 %matplotlib inline
704 plt.figure()
705 plt.plot(t, xphi)
706 plt.xlabel('t')
707 plt.ylabel('phi')
708 plt.grid()
709 plt.show()
710
711 plt.figure()
712 plt.plot(t, vphi)
713 plt.xlabel('t')
714 plt.ylabel('V')
715 plt.grid()
716 plt.show()
717
718 plt.figure()
719 plt.plot(t, aphi)
720 plt.xlabel('t')
721 plt.ylabel('d2phi/dt2')
722 plt.grid()
723 plt.show()
724
725 plt.figure()
726 plt.plot(t, xp)
727 plt.xlabel('t')
728 plt.ylabel('P')
729 plt.grid()
730 plt.show()
731
732 plt.figure()
733 plt.plot(t, vp)
734 plt.xlabel('t')
735 plt.ylabel('dP/dt')
736 plt.grid()
737 plt.show()
738
739 plt.figure()
740 plt.plot(t, ap)
741 plt.xlabel('t')
742 plt.ylabel('d2P/dt2')
743 plt.grid()
744 plt.show()
745
746 # In[RCSJ with polarisation]
747 @njit
748 def VV_RCSJ_p (R, C, gamma, m, a0, p0, I_c, I_max, num_current_steps, time_step, tspan, initc,
        avstart) :
749     # Equation constants
750     t = time_points(tspan, time_step)
751     damp = Q(I_c, R, C)
752     omegap = omega_p(I_c, C)
```

```
    dt = time_step
    a      = a0*(T-Tcurie)
    b      = ((-a)/(p0**2))

    # Velocity verlet constants
    A   = -1/damp
    B   = -omegap/I_c
    D   = cVphi/(omegap*m)
    E   = - gamma/(m*omegap)
    eta = 1 - E*dt/2
    xi  = 1 - A*dt/2 - B*D*dt**2/(4*eta)

    # Calculate number of current points as an integer
    I_data  = current_points(I_max, num_current_steps)
    I_steps = current_points(I_max, num_current_steps)[4].shape[0]
    CPR = np.zeros((6, I_steps))
    dpv = np.zeros(I_steps)
    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi, p = np.zeros((3, num_steps)), np.zeros((3, num_steps))
    # Assign initial conditions
    phi[0][0] = initc[0]
    phi[1][0] = initc[1]
    phi[2][0] = I_data[4][0]/I_c - np.sin(phi[0][0]) - phi[1][0]/damp - (omegap/I_c)*p[1][0]

    p[0][0] = initc[3]
    p[1][0] = initc[4]
    p[2][0] = (cVphi/(m*omegap))*phi[1][0] - (gamma/(m*omegap))*p[1][0] - (1/(m*omegap**2))*(a*p
    [0][0] + b*p[0][0]**3)


    for j in range(I_steps) :
        # Use the velocity verlet algorithm
        I = I_data[4][j]
        for i in range(num_steps-1) :
            phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
            p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2

            phi[1][i+1] = (1/xi)*( phi[1][i] + phi[2][i]*dt/2 + (B*dt/(2*eta))*(p[1][i] + p[2][i
    ]*dt/2) + (I/I_c - np.sin(phi[0][i+1]))*dt/2 + (B*dt**2/(4*eta))*(1/(m*omegap**2))*(-a*p[0][i
    +1] - b*p[0][i+1]**3) )
            p[1][i+1]   = (1/eta)*( p[1][i] + p[2][i]*dt/2 + (D*dt/2)*phi[1][i+1] + (1/(m*omegap
    **2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )

            phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp - (omegap/I_c)*p[1][i+1]
            p[2][i+1]   = (cVphi/(m*omegap))*phi[1][i+1] - (gamma/(m*omegap))*p[1][i+1] - (1/(m*
    omegap**2))*(a*p[0][i+1] + b*p[0][i+1]**3)

        phi_avg   = np.average(phi[0][avstart*tspan:])
        dphi_avg  = np.average(phi[1][avstart*tspan:])
        d2phi_avg = np.average(phi[2][avstart*tspan:])
        p_avg   = np.average(p[0][avstart*tspan:])
        dp_avg  = np.average(p[1][avstart*tspan:])
        d2p_avg = np.average(p[2][avstart*tspan:])
        V         = dphi_avg*cVphi*omegap
        CPR[0][j], CPR[1][j], CPR[2][j], CPR[3][j], CPR[4][j], CPR[5][j] = phi[0][-1], V, phi
    [2][-1], p[0][-1], dp_avg, d2p_avg
        phi[0][0],phi[1][0], p[0][0], p[1][0], p[2][0] = phi[0][-1], dphi_avg, p_avg, dp_avg, p
    [2][-1]

    return (I_data, CPR)

# In[]

```

```
ts = time.perf_counter_ns()
ic = np.array([0, 0, 0, 10, 0, 0])
RCSJp = VV_RCSJ_p(R, C, 1e-5, 1e-11, 1, 10, 1, 1.5, 1001, 0.01, 100, ic, 0.5)
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)
x = RCSJp[0][4]
yV = RCSJp[1][1]
yp = RCSJp[1][3]
ydp = RCSJp[1][4]

g = I_gap(yV, x)
print(g)

# In[Plot Data]
%matplotlib inline
plt.figure()
plt.plot(x, yV)
plt.title('RCSJ Hysteretic plot')
plt.xlabel('I')
plt.ylabel('V')
plt.grid()
plt.show()

plt.figure()
plt.plot(x, yp)
plt.title('polarisation')
plt.xlabel('I')
plt.ylabel('P')
plt.grid()
plt.show()

plt.figure()
plt.plot(x, ydp)
plt.title('polarisation speed')
plt.xlabel('I')
plt.ylabel('dP')
plt.grid()
plt.show()

# In[]
x1 = RCSJp[0][0]
x2 = RCSJp[0][1]
x3 = RCSJp[0][2]
x4 = RCSJp[0][3]
xp = RCSJp[0][7]
xn = RCSJp[0][6]

yV1 = RCSJp[1][1][:1001]
yV2 = RCSJp[1][1][1001:2002]
yV3 = RCSJp[1][1][2002:3003]
yV4 = RCSJp[1][1][3003:]
yV12 =  RCSJp[1][1][:2002]
yV34 = np.abs(RCSJp[1][1][2002:])



plt.figure()
plt.plot(x1, yV1)
plt.plot(x2, yV2)
plt.plot(x3, yV3)
plt.plot(x4, yV4)
plt.title('Hysteresis')
plt.xlabel('I')
```

```
plt.ylabel('P')
plt.grid()
plt.show()

%matplotlib qt5
plt.figure()
plt.plot(xp, yV12, 'g', label = 'pos I')
plt.plot(xp, yV34, 'r--', label = 'neg I')
plt.title('Voltage')
plt.xlabel('I')
plt.ylabel('V')
plt.legend(loc='best')
plt.grid()
plt.show()

# In[RCSJ with polarisation]
@njit
def VV_RCSJ_pm (R, C, gamma, m, a0, p0, I_c, I_max, num_current_steps, time_step, tspan, initc,
    avstart) :
    # Equation constants
    t = time_points(tspan, time_step)
    dt = time_step
    a      = a0*(T-Tcurie)
    b      = ((-a)/(p0**2))

    avs = np.array([avstart*tspan]).astype(np.int_)[0]

    # Calculate number of current points as an integer
    I_data  = current_points(I_max, num_current_steps)
    I_steps = current_points(I_max, num_current_steps)[4].shape[0]
    CPR = np.zeros((6, I_steps))
    CPR2 = np.zeros((6, I_steps))
    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi, p = np.zeros((3, num_steps)), np.zeros((3, num_steps))
    phi2, p2 = np.zeros((3, num_steps)), np.zeros((3, num_steps))

    if I_c == 0 :
        om =  np.sqrt(1/(cVphi*C))
        q0 = om*R*C
        # Assign initial conditions
        phi[0][0] = initc[0]
        phi[1][0] = initc[1]
        phi[2][0] = I_data[4][0] - phi[1][0]/q0 - om*p[1][0]

        p[0][0] = initc[3]
        p[1][0] = initc[4]
        p[2][0] = (cVphi/(m*om))*phi[1][0] - (gamma/(m*om))*p[1][0] - (1/(m*om**2))*(a*p[0][0] +
    b*p[0][0]**3)


        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            A1 = I/(cVphi*C)
            A2 = 1/(R*C)
            A3 = 1/(cVphi*C)
            B1 = cVphi/(m*om)
            B2 = a/(om**2)
            B3 = b/(m*om**2)
            B4 = gamma/(m*om)
            theta = 1 + B4
            sigma = 1 + (A2*dt/2) + (A3*B1*dt**2)/(4*theta)
            for i in range(num_steps-1) :
```

```
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2

                phi[1][i+1] = (1/sigma)*( phi[1][i] + phi[2][i]*dt/2 + A1*dt/2 - (A3*dt/(2*theta)
)*((p[1][i] + p[2][i]*dt/2)  + (dt/(2*m*om**2))*(-a*p[0][i+1] - b*p[0][i+1]**3)) )
                p[1][i+1]   = (1/theta)*( p[1][i] + p[2][i]*dt/2 + (B1*dt/2)*phi[1][i+1] + (dt
/(2*m*om**2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )

                phi[2][i+1] = A1 - A2*phi[1][i+1] - A3*p[1][i+1]
                p[2][i+1]   = B1*phi[1][i+1] - B2*p[0][i+1] - B3*p[0][i+1]**3 - B4*p[1][i+1]

            phi_avg   = np.average(phi[0][avs:])
            dphi_avg  = np.average(phi[1][avs:])
            d2phi_avg = np.average(phi[2][avs:])
            p_avg   = np.average(p[0][avs:])
            dp_avg  = np.average(p[1][avs:])
            d2p_avg = np.average(p[2][avs:])
            V         = dphi_avg*cVphi*om
            CPR[0][j], CPR[1][j], CPR[2][j], CPR[3][j], CPR[4][j], CPR[5][j] = phi[0][-1], V, phi
[2][-1], p[0][-1], dp_avg, d2p_avg
            phi[0][0],phi[1][0], p[0][0], p[1][0], p[2][0] = phi[0][-1], dphi_avg, p[0][-1],
dp_avg, p[2][-1]


            dphi_max  = np.max(phi[1])
            d2phi_avgf = np.average(phi[2][avs:])
            p_avgf   = np.average(p[0])
            dp_max  = np.max(p[1])
            d2p_avgf = np.average(p[2][avs:])
            Vm         = dphi_max*cVphi*om
            CPR2[0][j], CPR2[1][j], CPR2[2][j], CPR2[3][j], CPR2[4][j], CPR2[5][j] = phi[0][-1],
Vm, phi[2][-1], p[0][-1], dp_max, d2p_avgf
            phi2[0][0],phi2[1][0], p2[0][0], p2[1][0], p2[2][0] = phi2[0][-1], dphi_max, p_avgf,
dp_max, p[2][-1]


    else :
        damp = Q(I_c, R, C)
        omegap = omega_p(I_c, C)

        # Velocity verlet constants
        A   = -1/damp
        B   = -omegap/I_c
        D   = cVphi/(omegap*m)
        E   = - gamma/(m*omegap)
        eta = 1 - E*dt/2
        xi  = 1 - A*dt/2 - B*D*dt**2/(4*eta)

        # Assign initial conditions
        phi[0][0] = initc[0]
        phi[1][0] = initc[1]
        phi[2][0] = I_data[4][0]/I_c - np.sin(phi[0][0]) - phi[1][0]/damp - (omegap/I_c)*p[1][0]

        p[0][0] = initc[3]
        p[1][0] = initc[4]
        p[2][0] = (cVphi/(m*omegap))*phi[1][0] - (gamma/(m*omegap))*p[1][0] - (1/(m*omegap**2))*(
a*p[0][0] + b*p[0][0]**3)


        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            for i in range(num_steps-1) :
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2
```

```
997
998                 phi[1][i+1] = (1/xi)*( phi[1][i] + phi[2][i]*dt/2 + (B*dt/(2*eta))*(p[1][i] + p
    [2][i]*dt/2) + (I/I_c - np.sin(phi[0][i+1]))*dt/2 + (B*dt**2/(4*eta))*(1/(m*omegap**2))*(-a*p
    [0][i+1] - b*p[0][i+1]**3) )
999                 p[1][i+1]   = (1/eta)*( p[1][i] + p[2][i]*dt/2 + (D*dt/2)*phi[1][i+1] + (1/(m*
    omegap**2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )
1000
1001                phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp - (omegap/I_c)*p[1][
    i+1]
1002                p[2][i+1]   = (cVphi/(m*omegap))*phi[1][i+1] - (gamma/(m*omegap))*p[1][i+1] -
    (1/(m*omegap**2))*(a*p[0][i+1] + b*p[0][i+1]**3)
1003
1004            phi_avg   = np.average(phi[0][avs:])
1005            dphi_avg  = np.average(phi[1][avs:])
1006            d2phi_avg = np.average(phi[2][avs:])
1007            p_avg   = np.average(p[0][avs:])
1008            dp_avg  = np.average(p[1][avs:])
1009            d2p_avg = np.average(p[2][avs:])
1010            V         = dphi_avg*cVphi*omegap
1011            CPR[0][j], CPR[1][j], CPR[2][j], CPR[3][j], CPR[4][j], CPR[5][j] = phi[0][-1], V, phi
    [2][-1], p[0][-1], dp_avg, d2p_avg
1012            phi[0][0],phi[1][0], p[0][0], p[1][0], p[2][0] = phi[0][-1], dphi_avg, p[0][-1],
    dp_avg, p[2][-1]
1013
1014            dphi_max  = np.max(phi[1])
1015            d2phi_avgf = np.average(phi[2][avs:])
1016            p_avgf   = np.average(p[0])
1017            dp_max  = np.max(p[1])
1018            d2p_avgf = np.average(p[2][avs:])
1019            Vm         = dphi_max*cVphi*omegap
1020            CPR2[0][j], CPR2[1][j], CPR2[2][j], CPR2[3][j], CPR2[4][j], CPR2[5][j] = phi[0][-1],
    Vm, phi[2][-1], p[0][-1], dp_max, d2p_avgf
1021            phi2[0][0],phi2[1][0], p2[0][0], p2[1][0], p2[2][0] = phi2[0][-1], dphi_max, p_avgf,
    dp_max, p[2][-1]
1022
1023
1024    return (I_data, CPR, CPR2)
1025
1026 # In[]
1027
1028 ts = time.perf_counter_ns()
1029 ic = np.array([0, 0, 0, 1, 0, 0])
1030 RCSJp = VV_RCSJ_pm(R, C, 1e-5, 1e-11, 1, 1, 1, 1.5, 1501, 0.01, 100, ic, 0.5)
1031 tf = time.perf_counter_ns()
1032 tr = (tf - ts)*1e-9
1033 print(tr)
1034 x = RCSJp[0][4]
1035 yV = RCSJp[1][1]
1036 yp = RCSJp[1][3]
1037 ydp = RCSJp[1][4]
1038
1039 yV2 = RCSJp[2][1]
1040 yp2 = RCSJp[2][3]
1041 ydp2 = RCSJp[2][4]
1042
1043 g = I_gap(yV, x)
1044 g2 = I_gap(yV2, x)
1045 print(g, g2)
1046
1047 # In[Plot Data]
1048 %matplotlib qt5
1049 plt.figure()
1050 plt.plot(x, yV2, 'r--')
1051 plt.plot(x, yV)
1052 plt.title('RCSJ Hysteretic plot')
```

```
plt.xlabel('I')
plt.ylabel('V')
plt.grid()
plt.show()

plt.figure()
plt.plot(x, yp2, 'r--')
plt.plot(x, yp)
plt.title('polarisation')
plt.xlabel('I')
plt.ylabel('P')
plt.grid()
plt.show()

plt.figure()
plt.plot(x, ydp2, 'r--')
plt.plot(x, ydp)
plt.title('polarisation speed')
plt.xlabel('I')
plt.ylabel('dP')
plt.grid()
plt.show()

# In[]
x1 = RCSJp[0][0]
x2 = RCSJp[0][1]
x3 = RCSJp[0][2]
x4 = RCSJp[0][3]
xp = RCSJp[0][7]
xn = RCSJp[0][6]

yV1 = RCSJp[1][1][:1001]
yV2 = RCSJp[1][1][1001:2002]
yV3 = RCSJp[1][1][2002:3003]
yV4 = RCSJp[1][1][3003:]
yV12 =  RCSJp[1][1][:2002]
yV34 = np.abs(RCSJp[1][1][2002:])




plt.figure()
plt.plot(x1, yV1)
plt.plot(x2, yV2)
plt.plot(x3, yV3)
plt.plot(x4, yV4)
plt.title('Hysteresis')
plt.xlabel('I')
plt.ylabel('P')
plt.grid()
plt.show()

%matplotlib qt5
plt.figure()
plt.plot(xp, yV12, 'g', label = 'pos I')
plt.plot(xp, yV34, 'r--', label = 'neg I')
plt.title('Voltage')
plt.xlabel('I')
plt.ylabel('V')
plt.legend(loc='best')
plt.grid()
plt.show()

# In[RCSJ polarisation for different Ic including 0]

```

```
@njit
def VV_RCSJ_p2 ( R, C, gamma, m, a0, p0, I_c, I_max, num_current_steps, time_step, tspan, initc,
    avstart) :
    # Equation constants
    t = time_points(tspan, time_step)
    dt = time_step
    a      = a0*(T-Tcurie)
    b      = ((-a)/(p0**2))

    avs = np.array([avstart*tspan]).astype(np.int_)[0]

    # Calculate number of current points as an integer
    I_data  = current_points(I_max, num_current_steps)
    I_steps = current_points(I_max, num_current_steps)[4].shape[0]
    CPR = np.zeros((6, I_steps))
    # The number of points in t as an integer
    num_steps = np.array([t.shape[0]]).astype(np.int_)[0]
    # Create a zero array of shape (num_steps,3) for phi, dphi/dt and dphi^2/dt^2
    phi, p = np.zeros((3, num_steps)), np.zeros((3, num_steps))

    if I_c == 0 :
        om =  np.sqrt(1/(cVphi*C))
        q0 = om*R*C
        # Assign initial conditions
        phi[0][0] = initc[0]
        phi[1][0] = initc[1]
        phi[2][0] = I_data[4][0] - phi[1][0]/q0 - om*p[1][0]

        p[0][0] = initc[3]
        p[1][0] = initc[4]
        p[2][0] = (cVphi/(m*om))*phi[1][0] - (gamma/(m*om))*p[1][0] - (1/(m*om**2))*(a*p[0][0] +
    b*p[0][0]**3)


        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            A1 = I/(cVphi*C)
            A2 = 1/(R*C)
            A3 = 1/(cVphi*C)
            B1 = cVphi/(m*om)
            B2 = a/(om**2)
            B3 = b/(m*om**2)
            B4 = gamma/(m*om)
            theta = 1 + B4
            sigma = 1 + (A2*dt/2) + (A3*B1*dt**2)/(4*theta)
            for i in range(num_steps-1) :
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2

                phi[1][i+1] = (1/sigma)*( phi[1][i] + phi[2][i]*dt/2 + A1*dt/2 - (A3*dt/(2*theta)
    )*((p[1][i] + p[2][i]*dt/2)  + (dt/(2*m*om**2))*(-a*p[0][i+1] - b*p[0][i+1]**3)) )
                p[1][i+1]   = (1/theta)*( p[1][i] + p[2][i]*dt/2 + (B1*dt/2)*phi[1][i+1] + (dt
    /(2*m*om**2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )

                phi[2][i+1] = A1 - A2*phi[1][i+1] - A3*p[1][i+1]
                p[2][i+1]   = B1*phi[1][i+1] - B2*p[0][i+1] - B3*p[0][i+1]**3 - B4*p[1][i+1]

            phi_avg   = np.average(phi[0][avs:])
            dphi_avg  = np.average(phi[1][avs:])
            d2phi_avg = np.average(phi[2][avs:])
            p_avg   = np.average(p[0][avs:])
            dp_avg  = np.average(p[1][avs:])
            d2p_avg = np.average(p[2][avs:])
            V         = dphi_avg*cVphi*om
```

```
            CPR[0][j], CPR[1][j], CPR[2][j], CPR[3][j], CPR[4][j], CPR[5][j] = phi[0][-1], V, phi[2][-1], p[0][-1], dp_avg, d2p_avg
            phi[0][0],phi[1][0], p[0][0], p[1][0], p[2][0] = phi[0][-1], dphi_avg, p[0][-1], dp_avg, p[2][-1]

    else :
        damp = Q(I_c, R, C)
        omegap = omega_p(I_c, C)

        # Velocity verlet constants
        A   = -1/damp
        B   = -omegap/I_c
        D   = cVphi/(omegap*m)
        E   = - gamma/(m*omegap)
        eta = 1 - E*dt/2
        xi  = 1 - A*dt/2 - B*D*dt**2/(4*eta)

        # Assign initial conditions
        phi[0][0] = initc[0]
        phi[1][0] = initc[1]
        phi[2][0] = I_data[4][0]/I_c - np.sin(phi[0][0]) - phi[1][0]/damp - (omegap/I_c)*p[1][0]

        p[0][0] = initc[3]
        p[1][0] = initc[4]
        p[2][0] = (cVphi/(m*omegap))*phi[1][0] - (gamma/(m*omegap))*p[1][0] - (1/(m*omegap**2))*(a*p[0][0] + b*p[0][0]**3)


        for j in range(I_steps) :
            # Use the velocity verlet algorithm
            I = I_data[4][j]
            for i in range(num_steps-1) :
                phi[0][i+1] = phi[0][i] + phi[1][i]*dt + phi[2][i]*dt**2/2
                p[0][i+1]   = p[0][i] + p[1][i]*dt + p[2][i]*dt**2/2

                phi[1][i+1] = (1/xi)*( phi[1][i] + phi[2][i]*dt/2 + (B*dt/(2*eta))*(p[1][i] + p[2][i]*dt/2) + (I/I_c - np.sin(phi[0][i+1]))*dt/2 + (B*dt**2/(4*eta))*(1/(m*omegap**2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )
                p[1][i+1]   = (1/eta)*( p[1][i] + p[2][i]*dt/2 + (D*dt/2)*phi[1][i+1] + (1/(m*omegap**2))*(-a*p[0][i+1] - b*p[0][i+1]**3) )

                phi[2][i+1] = I/I_c - np.sin(phi[0][i+1]) - phi[1][i+1]/damp - (omegap/I_c)*p[1][i+1]
                p[2][i+1]   = (cVphi/(m*omegap))*phi[1][i+1] - (gamma/(m*omegap))*p[1][i+1] - (1/(m*omegap**2))*(a*p[0][i+1] + b*p[0][i+1]**3)

            phi_avg   = np.average(phi[0][avs:])
            dphi_avg  = np.average(phi[1][avs:])
            d2phi_avg = np.average(phi[2][avs:])
            p_avg   = np.average(p[0][avs:])
            dp_avg  = np.average(p[1][avs:])
            d2p_avg = np.average(p[2][avs:])
            V         = dphi_avg*cVphi*omegap
            CPR[0][j], CPR[1][j], CPR[2][j], CPR[3][j], CPR[4][j], CPR[5][j] = phi[0][-1], V, phi[2][-1], p[0][-1], dp_avg, d2p_avg
            phi[0][0],phi[1][0], p[0][0], p[1][0], p[2][0] = phi[0][-1], dphi_avg, p[0][-1], dp_avg, p[2][-1]

    return (I_data, CPR)
# In[]
ts = time.perf_counter_ns()
ic = np.array([0, 0, 0, 1, 0, 0])
RCSJp = VV_RCSJ_p2(R, C, 1e-5, 1e-11, 1, 1, 1, 1.5, 1001, 0.01, 100, ic, 0.5)
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
```

```
print(tr)
x = RCSJp[0][4]
yV = RCSJp[1][1]
yp = RCSJp[1][3]

cpr = RCSJp[1]

# In[Plot Data]
%matplotlib inline
plt.figure()
plt.plot(x, yV)
plt.title('RCSJ Hysteretic plot')
plt.xlabel('I')
plt.ylabel('V')
plt.grid()
plt.show()

plt.figure()
plt.plot(x, yp)
plt.title('polarisation')
plt.xlabel('I')
plt.ylabel('P')
plt.grid()
plt.show()

# In[]
x1 = RCSJp[0][0]
x2 = RCSJp[0][1]
x3 = RCSJp[0][2]
x4 = RCSJp[0][3]
xp = RCSJp[0][7]
xn = RCSJp[0][6]

oc = 1001

yV1 = RCSJp[1][1][:oc]
yV2 = RCSJp[1][1][oc:2*oc]
yV3 = RCSJp[1][1][2*oc:3*oc]
yV4 = RCSJp[1][1][3*oc:]
yV12 =  RCSJp[1][1][:2*oc]
yV34 = np.abs(RCSJp[1][1][2*oc:])

plt.figure()
plt.plot(x1, yV1)
plt.plot(x2, yV2)
plt.plot(x3, yV3)
plt.plot(x4, yV4)
plt.title('Hysteresis')
plt.xlabel('I')
plt.ylabel('P')
plt.grid()
plt.show()

%matplotlib qt5
plt.figure()
plt.plot(xp, yV12, 'g', label = 'pos I')
plt.plot(xp, yV34, 'r--', label = 'neg I')
plt.title('Voltage')
plt.xlabel('I')
plt.ylabel('V')
plt.legend(loc='best')
plt.grid()
plt.show()

# In[RCSJ with pol and flux]
```

```
@njit
def RCSJ_pf(flux, R, C, gamma, m, a0, p0, I_0, I_max, num_current_steps, time_step, tspan, initc,
     avstart) :

    fsteps = np.array([flux.shape[0]]).astype(np.int_)[0]
    I_data = current_points(I_max, num_current_steps)
    I_stepsf = I_data[4].shape[0]
    VIf = np.zeros((fsteps, 6, I_stepsf))
    Ics = np.zeros(fsteps)

    for i in range(fsteps) :
        I_c = I_0*np.abs(np.round(np.sinc(flux[i]), 16))
        Ics[i] = I_c
        initc[0] = 2*pi*flux[i]
        data = VV_RCSJ_pm( R, C, gamma, m, a0, p0, I_c, I_max, num_current_steps, time_step,
    tspan, initc, avstart)
        VIf[i] = data[2]
    return (I_data, VIf, Ics)

# In[]
ts = time.perf_counter_ns()
fl = np.linspace(0, 1, num = 501)
ic = np.array([0, 0, 0, 1, 0, 0])
pfl_data = RCSJ_pf(fl, R, C, 1e-5, 1e-11, 1, 1, 1, 1.5, 15001, 0.01, 100, ic, 0.5)
tf = time.perf_counter_ns()
tr = (tf - ts)*1e-9
print(tr)

x = pfl_data[0]
y = pfl_data[1]

# In[]
%matplotlib inline
Icdata = pfl_data[2]
thIc = np.abs(np.sinc(fl))
plt.figure()
plt.plot(fl, Icdata)
plt.plot(fl, thIc, 'r--')
plt.grid()

# In[]
%matplotlib inline
times = np.zeros((len(Icdata)))
oms = np.zeros((len(fl)))
for i in range(len(Icdata)) :
    Ic = Icdata[i]
    if Ic == 0 :
        w = omega_p(1, C)
        t = (1/w)*100
        oms[i] = w
    else :
        w = omega_p(Ic, C)
        t = (1/w)*100
        oms[i] = w
    times[i] = t
plt.figure()
plt.plot(fl, times, '-')
#plt.plot(fl, oms, '-')
plt.xlabel('phi/phi_0')
plt.ylabel('time')
plt.grid()
# In[]
xpfl = x[4]

x1 = x[0]
```

```
x2 = x[1]
x3 = x[2]
x4 = x[3]
xp = x[7]
xn = x[6]

fc = 15001
Igap = np.zeros(501)

%matplotlib inline

for i in range(501) :

    ydata = y[i]
    yVp = ydata[1][:2*fc]
    yVn = np.abs(ydata[1][2*fc:])
    yP = ydata[3]
    ydP = ydata[4]

    m = np.round(fl[i], 3)
    g = np.round(I_gap(ydata[1], xpfl), 7)

    Igap[i] = g[1]

#    plt.figure()
#    plt.plot(xp, yVp, 'g', label = 'pos I')
#    plt.plot(xp, yVn, 'r--', label = 'neg I')
#    plt.title(str(g) + str(m))
#    plt.xlabel('I')
#    plt.ylabel('V')
#    plt.legend(loc='best')
#    plt.grid()
#    plt.show()

# In[]
%matplotlib qt5

rcParams.update({'font.size': 22})

plt.figure(figsize=[15, 10], dpi = 150)
plt.plot(fl, Igap, '-')
plt.xlabel('$\Phi/\Phi_0$')
plt.ylabel('I gap')
plt.grid()
plt.show()
plt.savefig('C:/Users/Keane/Desktop/BEP/Figures/RCSJ Igap versua flux.png')

# In[]
```