The Josephson effect

Johanne B. Tjernshaugen

Praktisk informasjon

Innleveringsfrist: Fredag 7. mars 2015, kl. 23:59.
Innleveringsplattform: Inspera.
Språk: Dere kan velge om dere vil svare på engelsk eller norsk.
Vurderingsansvarlig: Johanne B. Tjernshaugen, mail: `johanne.b.tjernshaugen@ntnu.no`
Innleveringsformat: Jupyter Notebook (lever kun én fil).

Notebooken skal være kjørt ved innlevering. For å sjekke om den er kjørt kan dere laste ned den endelige notebooken. Dere skal referere til oppgavenummer (for eksempel **2b**) i besvarelsen. Oppgavene skal besvares i riktig rekkefølge. Det er ikke nødvendig å skrive introduksjon, metode osv. som man vanligvis skriver i en labrapport. Svar kun på det oppgavene spør om.

Før dere leverer bør dere lese nøye gjennom besvarelsen. Bruk gjerne stavekontroll.

Alle plott skal forklares/diskuteres, og alle akser skal ha navn ("labels") med korrekte fysiske enheter. Koden deres skal være enkel å lese og forståelig med informative variabelnavn. Bruk gjerne kommentarer. Godt dokumentert og strukturert kode vil trekke besvarelsen i positiv retning. Kommentarer og dokumentasjon er spesielt vikitg i deloppgavene som kun ber dere skrive en funksjon. Kjøretiden på koden deres vil variere avhengig av hvordan dere løser oppgavene, men vil ikke påvirke endelig karakter utover at rask/lur/godt strukturert kode vil telle i positiv retning. Det er fint mulig å få full uttelling med tregere kode.

Karakteren blir satt ut fra hvor riktige resultatene er og diskusjonen av resultatene. Resultater som åpenbart er feil vil ikke gi maksimal poengsum, men diskusjonen av resultatene vil kunne trekke besvarelsen i positiv retning. Hvis dere forstår at svarene deres er feil eller ufysiske vil dere få en høyere poengsum hvis dere forklarer hva som er feil og hvorfor det er feil, ettersom dette viser fysisk intuisjon og forståelse.

Det er lov å ta utgangspunkt i kodesnutter fra forelesningsmaterialet der dere anser dette som hensiktsmessig.

Introduction

In 1911, Heike Onnes discovered that when mercury was cooled below 4.2 K, its electrical resistance dropped to zero. This remarkable phenomenon is known as superconductivity. At this time, quantum physics had just been born and the microscopic mechanism behind superconductivity remained unclear for almost 50 years. In 1957, John Bardeen, Leon Cooper and Robert Schrieffer proposed that electrons in a metal can interact attractively at low temperatures due to lattice vibrations (”phonons”) and that this attraction can be stronger than the usual Coulomb repulsion between the negatively charged electrons. Hence, two electrons can pair up in a bound state known as a Cooper pair, and this causes superconductivity. Bardeen, Cooper and Schrieffer received the Nobel Prize in 1972 for this theory, now known as the BCS theory. As a result of the electrons forming Cooper pairs, a gap shows up around the Fermi energy in the electronic density of states. This is shown in Fig. 1. The density of states is the number of available states for electrons with a given energy. The density of states is an important concept for describing the properties of solids, such as their abilities to conduct a current. The Fermi energy is the energy difference between the highest and lowest occupied electronic states at zero temperature. Generally, materials with a high density of states near the Fermi energy are good conductors. Superconductors do not have electrons near the Fermi energy, but can nevertheless conduct currents because the Cooper pairs have energies close to the Fermi energy. The gap in the

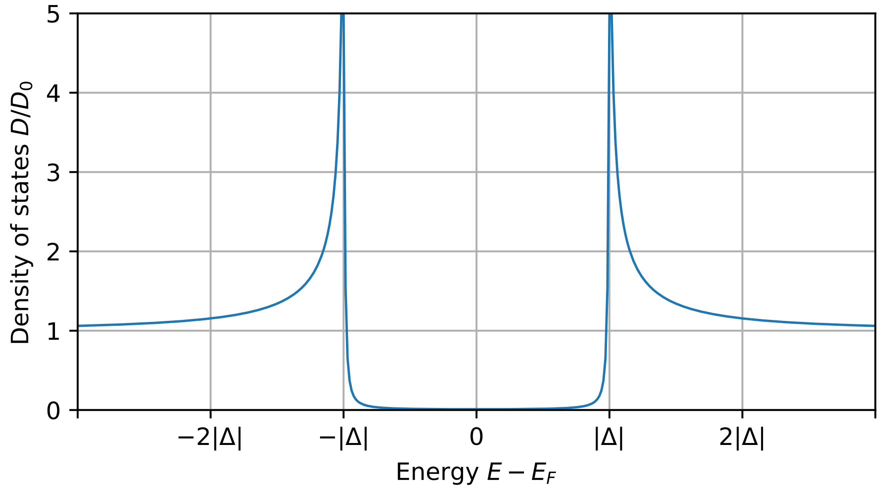

Figure 1: Density of states D in a superconductor as a function of energy $E - E_F$. The energy is measured relative to the Fermi energy E_F, and D is normalized on the density of states D_0 in a non-superconducting metal. There is a gap in the density of states for energies between $-|\Delta|$ and $|\Delta|$. This means that single electrons cannot exist in the superconductor if their energy is close to the Fermi energy.

density of states is related to the superconducting order parameter, which is another important concept in condensed matter physics. The order parameter characterizes order. For superconductors, ”order” means that there exist Cooper pairs and the gap $|\Delta|$ is nonzero. ”No order” means that there is no superconductivity and no Cooper pairs, and there is no gap in the density of states ($|\Delta| = 0$). The superconducting order parameter Δ is in general a complex number, and it is defined as follows:

$$\Delta = |\Delta| e^{i\phi}. \tag{1}$$

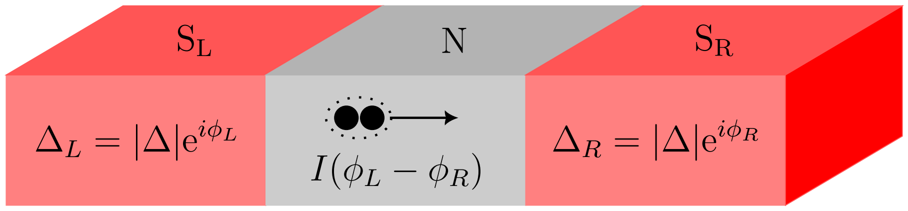
$\mathrm{S_L}$
N
$\mathrm{S_R}$
$\Delta_L = |\Delta|\mathrm{e}^{i\phi_L}$
$I(\phi_L - \phi_R)$
$\Delta_R = |\Delta|\mathrm{e}^{i\phi_R}$

Figure 2: A Josephson junction. Two superconductors $\mathrm{S_L}$ and $\mathrm{S_R}$ are separated by a non-superconducting (”normal”) metal N. The gap $|\Delta|$ is the same in both superconductors, but the phases ϕ_L and ϕ_R may be different. Depending on the phase difference, a supercurrent $I(\phi_L - \phi_R)$ carried by Cooper pairs may flow through N.

Here, $|\Delta|$ and ϕ are real numbers called the superconducting gap and the superconducting phase, respectively. The phase of one single superconductor has no physical meaning and can be set to zero, such that the order parameter is identical to the gap. However, as we will see in this project, a phase difference between two superconductors separated by a barrier (a Josephson junction, see Fig. 2) causes a current to flow between them. This is called the Josephson effect after its discoverer Brian Josephson, who was awarded the Nobel Prize in 1973 for his prediction. The current is carried by Cooper pairs and thus it is a supercurrent with zero resistivity. The Josephson junction is a key element in superconducting electronics. It is used in a large variety of devices for many different applications, for example in SQUIDs (superconducting quantum interference devices) and transmon qubits. SQUIDs are very sensitive detectors of magnetic fields and can be used to detect very tiny variations in the magnetic fields of the Earth and also of the human body. The transmon is a type of qubit for use in quantum computers.

In this project, the goal is to calculate the supercurrent between two superconductors separated by a non-superconducting (”normal”) metal. You will do this by solving the one-dimensional Usadel equation, named after Klaus Usadel, numerically in the normal metal. The Usadel equation is derived from a quantum model by assuming that the Fermi energy is the largest energy in the system, and that impurities in the material cause the electrons’ movement to be diffusive-like. The predictions from the Usadel equation are known to compare well with experiments on superconductors on the nano- to micrometer scale.

The Usadel equation is solved together with boundary conditions at each end of the normal metal, and the goal is therefore to solve a boundary value problem. In exercise 1 you will solve a simple boundary value problem. In exercise 2 you will solve the Usadel equation using a built-in boundary value problem solver from `scipy`.

1 Solving a simple boundary value problem

The shooting method

The goal of exercise 1 is to solve a simple boundary value problem. Consider the equation

$$\frac{d^2y(x)}{dx^2} = f(x, y)$$

on the interval $x \in [x_{init}, x_{end}]$. At the edges, the solution is fixed to

$$y(x_{init}) = A \text{ and } y(x_{end}) = B.$$

This is a *boundary* value problem. Numerically it is easier to solve *initial* value problems where the derivative $y'(x_{init}) = b$ is known, but $y(x_{end})$ is not. Consequently, the simplest method for solving the boundary value problem is to solve the initial value problem for different b until the boundary condition at x_{end} is met. Successive guesses for b are calculated using some root-finding algorithm. This is known as the "shooting" method. You can think of this as trying to hit a target with a bow and arrow: the arrowhead is held a constant height above ground, and you aim by varying the angle. This is illustrated in Fig. 3.

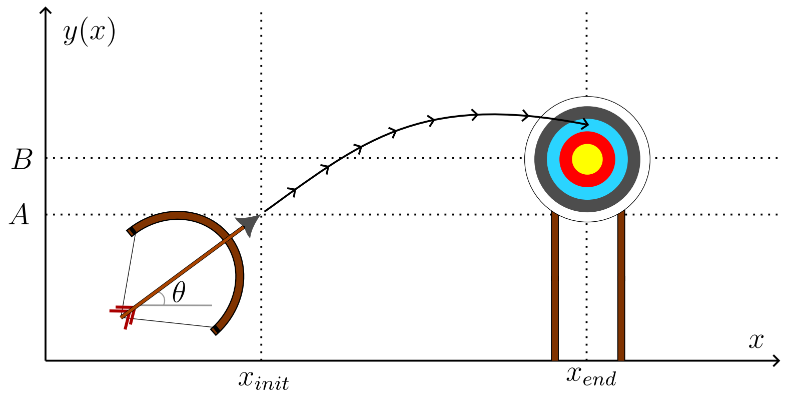

Figure 3: An illustration of the "shooting" method. $y(x)$ is the trajectory of the arrowhead. The arrow is shot from the position where the arrowhead has coordinates (x_{init}, A) and the goal is to hit the target at height B at position x_{end}. This is done by choosing the angle θ correctly. The angle is related to the derivative $y'(x_{init}) = b$ by $\theta = \text{atan}(b)$.

In exercises 1a)-1d), you will solve an initial value problem. You will use a Runge-Kutta method of order 3(2) as described below. In exercise 1e), you will write a function that finds the root of a function. Finally, in exercise 1g) you will solve a boundary value problem with the shooting method, and in exercise 1h) you will solve the same boundary value problem with a built-in `scipy` function.

Solving an initial value problem

In 1989, Lawrence Shampine and Przemyslaw Bogacki published a paper on a Runge-Kutta method of order 3(2) for solving initial value problems of the type

$$\frac{d^2y}{dx^2} = f(x, y), \quad y(x_{init}) = A, \quad y'(x_{init}) = b, \quad x \in [x_{init}, x_{end}]. \tag{2}$$

The method is an adaptive method that chooses successive x-values (the "time step") automatically.

Equation (2) can be rewritten to a first-order differential equation by defining the vector $\vec{y}(x) := (y(x), y'(x)) := (y_1(x), y_2(x))$:

$$\frac{d\vec{y}}{dx} = \vec{f}(x, \vec{y}), \quad \vec{y}(x_{init}) = (A, b), \quad x \in [x_{init}, x_{end}]. \tag{3}$$

Here, $f_1(x, \vec{y}) = y_2(x)$ and $f_2(x, \vec{y}) = f(x, y_1(x))$. One step of the method can be formulated as

$$\begin{aligned}
\vec{k}_1 &= \vec{f}(x_n, \vec{y}_n) \\
\vec{k}_2 &= \vec{f}(x_n + \frac{1}{2}h_n, \vec{y}_n + \frac{1}{2}h_n\vec{k}_1) \\
\vec{k}_3 &= \vec{f}(x_n + \frac{3}{4}h_n, \vec{y}_n + \frac{3}{4}h_n\vec{k}_2) \\
\vec{y}_{n+1} &= \vec{y}_n + \frac{1}{9}h_n(2\vec{k}_1 + 3\vec{k}_2 + 4\vec{k}_3) \\
\vec{k}_4 &= \vec{f}(x_n + h_n, \vec{y}_{n+1}) \\
\vec{z}_{n+1} &= \vec{y}_n + \frac{1}{24}h_n(7\vec{k}_1 + 6\vec{k}_2 + 8\vec{k}_3 + 3\vec{k}_4)
\end{aligned}$$

This method can be implemented by writing a function that takes the following input:

- `x_init`: The starting point
- `x_end`: The endpoint
- `y_init`: The initial value at `x_init`
- `f`: A function as described below
- `h0`: An approximate first step size
- `tol`: A tolerance for the local error

The function returns a `numpy` array `x=[x_init,x1,x2,...,x_end]` where `x1,x2...` are chosen by the function. It also returns a 2D `numpy` matrix where row `n` is the solution at position `x_n`. Additionally, it returns useful information such as the step length at each step and the total number of steps taken. The input function `f` takes two input arguments `x` and `y`, where `x` is a real number and `y` is a `numpy` array with dimensions equal to the dimension of the problem. It returns a `numpy` array of the same dimension as `y` containing `f(x,y)`.

When we try taking one step with the method, we first calculate $\vec{y}_{n+1}$ and $\vec{z}_{n+1}$ from the formulas given earlier, and then we create a scalar error estimate:

$$\text{est}_{n+1} = ||\vec{y}_{n+1} - \vec{z}_{n+1}||.$$

Additionally, there is a formula for adjusting the step length from one step to another. We set

$$h_{new} = \alpha \cdot h_n \cdot \left(\frac{\text{tol}}{\text{est}}\right)^{1/3}.$$

α is a pessimist factor. If it is chosen too small the steps are smaller than necessary, and one uses many steps for solving the problem. On the other hand, if α is chosen too large many of the steps may be discarded, and the calculation time grows as many of the steps must be recalculated. The algorithm can be described as follows:

```
• Given $x_{init}, x_{end}, \vec{y}_{init}, h_0, tol, \vec{f}, \alpha$ as input
• $h = h_0$, $n = 0$
• $\vec{k}_1 = \vec{f}(x_{init}, \vec{y}_{init})$
• while $x_{end} - x_n > 0$
    – $h = \min(h, x_{end} - x_n)$
    – Calculate $\vec{k}_2, \vec{k}_3$ and then $\vec{y}_{n+1}$
    – $x_{n+1} = x_n + h$
    – Calculate $\vec{k}_4$ and then $\vec{z}_{n+1}$
    – $est = \|\vec{y}_{n+1} - \vec{z}_{n+1}\|$
    – if $est < tol$ (accept step)
        ∗ $n = n + 1$
        ∗ $k_1 = k_4$
    – Calculate $h_{new}$ and let $h = h_{new}$.
```

Note that the same formula for updating the step length h is used both when the step is accepted and discarded.

Exercise 1a) Solve analytically (using pen and paper) the initial value problem

$$\frac{d^2y}{dx^2} = -4 \cdot \sin(2x), \quad y(0) = 0, \quad y'(0) = 2.$$

Exercise 1b) Rewrite the second order differential equation for the scalar $y(x)$ in exercise 1a) to a first order differential equation for the vector $\vec{y}(x) = \begin{pmatrix} y(x) \\ y'(x) \end{pmatrix}$. Write down the initial condition for $\vec{y}$.

Exercise 1c) Write the code described above and solve the equations you found in 1b) with your code. Let $x \in [0, 2\pi]$, $\alpha = 0.8$ and `tol`$= 10^{-7}$. Plot $y(x)$ and $y'(x)$ as a function of x calculated by your code. Plot how the step length varies as a function of x, and explain qualitatively why the step length varies as a function of x.

Exercise 1d) The parameters α and `tol` are input parameters to the function, and they both affect the running time of the function. Make the following two plots:

- Plot how the actual error depends on the tolerance `tol`. Calculate the error by comparing the result from your code with the exact expression you found in exercise 1a).
- Plot the number of time steps (both accepted and discarded) as a function of the parameter α.

As always, remember to comment on and discuss the plots.

Finding the root of a function

The secant method is one of the simplest methods for finding the root of a function g, that is, the value z for which $g(z) = 0$. This is how the method works:

- Choose two initial guesses z_0 and z_1 for z.
- Proceed by calculating
$$z_n = \frac{z_{n-2}g(z_{n-1}) - z_{n-1}g(z_{n-2})}{g(z_{n-1}) - g(z_{n-2})}. \tag{4}$$
- Repeat until the difference $|z_{n-1} - z_n| < tol$ for some tolerance tol.

Exercise 1e) Write a function that solves a root finding problem with the secant method. The function takes the following input: a function `g`, two initial guesses `z0` and `z1`, and a tolerance `tol`. It returns the root of `g`. Test the function by finding the root of $g(z) = z + \sin(z) + \cos(z)$. You choose the initial guesses and the tolerance. Confirm that the root you find actually is the root of g.

Solving a boundary value problem

We are now ready to solve the boundary value problem

$$\frac{d^2y}{dx^2} = -4 \cdot \sin(2x), \quad y(0) = 0, \quad y(2\pi) = 0. \tag{5}$$

This problem has exactly the same solution as in exercise 1a), but the goal is now to make a function that solves the problem without knowing the initial value $y'(x_{init})$. We can do this by introducing the unknown variable b and solving the initial value problem

$$\frac{d^2y}{dx^2} = -4 \cdot \sin(2x), \quad y(0) = 0, \quad y'(0) = b.$$

The solution $y_b(x)$ to this problem depends on b, and we want to find the b-value that gives

$$y_b(2\pi) = 0. \tag{6}$$

When using the secant method to solve Eq. (6), one needs two initial guesses b_0 and b_1 for b.

Exercise 1f) Use your initial value solver from exercise 1c) and your root finder from exercise 1e) to solve the boundary value problem given in eq. (5). Make a plot showing $y(x)$ as a function of x calculated by your code for each iteration of the root finder.

Exercise 1g) Solve the following boundary value problem using your initial value solver and root finder:

$$\frac{d^2y}{dx^2} = y(x) + \sin(x), \quad y(0) = y(12) = 0. \tag{7}$$

Plot the solution $y(x)$ on the interval $x \in [0, 12]$.

Exercise 1h) Use the `scipy` function `solve_bvp` to solve eq. (7). Compare the result to your results from exercise 1g):

- Plot the solution $y(x)$ on the interval $x \in [0, 12]$.
- Plot the difference between your solution for $y(x)$ and the solution from `solve_bvp` as a function of x.
- Considering the running time of your functions vs. `solve_bvp`, which solver would you prefer to use?

Now that we have an idea of how to solve a boundary value problem numerically, we turn to the physical problem that we want to solve: a normal metal interfaced with two superconductors. For the rest of this project, it is highly recommended to use built-in functions from e.g. `numpy` and `scipy` whenever applicable. You are NOT supposed to write your own boundary value problem solver for exercise 2; use `solve_bvp` from `scipy`.

2 The Usadel equations

The goal of this exercise is to solve the Usadel equations inside the normal region of the Josephson junction in Fig. 2. We want to calculate the Green function $\hat{g}$, which is a 4×4 matrix containing information such as the density of states and the current. The Green function depends both on position x' and energy ε'. In numerical calculations, it is convenient to use dimensionless quantities. The dimensionless position $x = x'/\xi$ is normalized against the superconducting coherence length ξ, which is a measure of the size of the Cooper pairs. The dimensionless energy ε is normalized against the superconducting gap and relative to the Fermi energy, $\varepsilon = (E - E_F)/|\Delta|$.

Before we go on, we introduce some notation. All variables in "normal" text, such as x, ε and δ are numbers. To make clear the dimensions of matrices, all 4×4 matrices have a hat (for example $\hat{g}, \hat{\rho}_3$) and 2×2 matrices are written in bold font (for example $\boldsymbol{\gamma}, \tilde{\boldsymbol{\omega}}$). For short, we write the derivative as $\partial_x \equiv \frac{d}{dx}$.

The Usadel equation can be written as four first-order differential equations consisting of 2×2 matrices. The goal is to find the matrices $\boldsymbol{\gamma}(x, \varepsilon), \tilde{\boldsymbol{\gamma}}(x, \varepsilon), \boldsymbol{\omega}(x, \varepsilon)$ and $\tilde{\boldsymbol{\omega}}(x, \varepsilon)$, from which we can calculate the Green function:

$$\hat{g}(x, \varepsilon) = \begin{pmatrix} 2\boldsymbol{N}(x, \varepsilon) - \boldsymbol{I} & 2\boldsymbol{N}(x, \varepsilon)\boldsymbol{\gamma}(x, \varepsilon) \\ -2\tilde{\boldsymbol{N}}(x, \varepsilon)\tilde{\boldsymbol{\gamma}}(x, \varepsilon) & -2\tilde{\boldsymbol{N}}(x, \varepsilon) + \boldsymbol{I} \end{pmatrix}. \tag{8}$$

Here, $\boldsymbol{I}$ is the 2×2 identity matrix, $\boldsymbol{N}(x,\varepsilon) = [\boldsymbol{I} - \boldsymbol{\gamma}(x,\varepsilon)\tilde{\boldsymbol{\gamma}}(x,\varepsilon)]^{-1}$ and $\tilde{\boldsymbol{N}}(x,\varepsilon) = [\boldsymbol{I} - \tilde{\boldsymbol{\gamma}}(x,\varepsilon)\boldsymbol{\gamma}(x,\varepsilon)]^{-1}$. The matrices $\boldsymbol{\gamma}(x,\varepsilon)$ and $\tilde{\boldsymbol{\gamma}}(x,\varepsilon)$ are known as *Riccati* matrices, and $\boldsymbol{\omega}(x,\varepsilon)$ and $\tilde{\boldsymbol{\omega}}(x,\varepsilon)$ are their derivatives, respectively. $\boldsymbol{\gamma}(x,\varepsilon)$ and $\tilde{\boldsymbol{\gamma}}(x,\varepsilon)$ are related by $\tilde{\boldsymbol{\gamma}}(x,\varepsilon) = \boldsymbol{\gamma}^*(x,-\varepsilon)$ where * means complex conjugation, but we treat them as independent variables because it allows us to solve the Usadel equations for $\varepsilon \geq 0$ only.

The Usadel equations inside the normal region are

$$\partial_x \boldsymbol{\gamma}(x,\varepsilon) = \boldsymbol{\omega}(x,\varepsilon), \tag{9}$$

$$\partial_x \tilde{\boldsymbol{\gamma}}(x,\varepsilon) = \tilde{\boldsymbol{\omega}}(x,\varepsilon), \tag{10}$$

$$\partial_x \boldsymbol{\omega}(x,\varepsilon) = -2i(\varepsilon + i\delta)\boldsymbol{\gamma}(x,\varepsilon) - 2\boldsymbol{\omega}(x,\varepsilon)\tilde{\boldsymbol{N}}(x,\varepsilon)\tilde{\boldsymbol{\gamma}}(x,\varepsilon)\boldsymbol{\omega}(x,\varepsilon), \tag{11}$$

$$\partial_x \tilde{\boldsymbol{\omega}}(x,\varepsilon) = -2i(\varepsilon + i\delta)\tilde{\boldsymbol{\gamma}}(x,\varepsilon) - 2\tilde{\boldsymbol{\omega}}(x,\varepsilon)\boldsymbol{N}(x,\varepsilon)\boldsymbol{\gamma}(x,\varepsilon)\tilde{\boldsymbol{\omega}}(x,\varepsilon). \tag{12}$$

These are the equations we want to solve. The parameter δ describes inelastic scattering, and we set it to $\delta = 0.01$. Be aware that matrix multiplication must be performed in the correct order!

Since we want to solve the equations on the interval $x \in [0, l]$, where l is the normalized length of the normal region, we need boundary conditions at the left ($x = 0$) and right ($x = l$) side. In the Josephson junction in Fig. 2, the normal metal is interfaced with two superconductors. In general, the normal metal could be interfaced with other materials, for example another normal metal or a magnet. Therefore, we generally take the Riccati matrices in the material on the left side to be $\boldsymbol{\gamma}_L(\varepsilon)$ and $\tilde{\boldsymbol{\gamma}}_L(\varepsilon)$, and on the right side $\boldsymbol{\gamma}_R(\varepsilon)$ and $\tilde{\boldsymbol{\gamma}}_R(\varepsilon)$. We use quantum mechanical tunneling boundary conditions, which on the left side reads:

$$\boldsymbol{\omega}(0,\varepsilon) + \frac{1}{\zeta l}[\boldsymbol{I} - \boldsymbol{\gamma}(0,\varepsilon)\tilde{\boldsymbol{\gamma}}_L(\varepsilon)] \cdot \boldsymbol{N}_L(\varepsilon) \cdot [\boldsymbol{\gamma}_L(\varepsilon) - \boldsymbol{\gamma}(0,\varepsilon)] = \boldsymbol{0}, \tag{13}$$

$$\tilde{\boldsymbol{\omega}}(0,\varepsilon) + \frac{1}{\zeta l}[\boldsymbol{I} - \tilde{\boldsymbol{\gamma}}(0,\varepsilon)\boldsymbol{\gamma}_L(\varepsilon)] \cdot \tilde{\boldsymbol{N}}_L(\varepsilon) \cdot [\tilde{\boldsymbol{\gamma}}_L(\varepsilon) - \tilde{\boldsymbol{\gamma}}(0,\varepsilon)] = \boldsymbol{0}. \tag{14}$$

Here, ζ is an interface parameter that describes the ratio of the interface resistance to the bulk resistance, and we set it to $\zeta = 3$. Also, we have $\boldsymbol{N}_L(\varepsilon) = [\boldsymbol{I} - \boldsymbol{\gamma}_L(\varepsilon)\tilde{\boldsymbol{\gamma}}_L(\varepsilon)]^{-1}$ and $\tilde{\boldsymbol{N}}_L(\varepsilon) = [\boldsymbol{I} - \tilde{\boldsymbol{\gamma}}_L(\varepsilon)\boldsymbol{\gamma}_L(\varepsilon)]^{-1}$. The boundary conditions on the right interface have almost the same form as on the left, but note that two of the plus signs have been replaced by minus signs:

$$\boldsymbol{\omega}(l,\varepsilon) - \frac{1}{\zeta l}[\boldsymbol{I} - \boldsymbol{\gamma}(l,\varepsilon)\tilde{\boldsymbol{\gamma}}_R(\varepsilon)] \cdot \boldsymbol{N}_R(\varepsilon) \cdot [\boldsymbol{\gamma}_R(\varepsilon) - \boldsymbol{\gamma}(l,\varepsilon)] = \boldsymbol{0}, \tag{15}$$

$$\tilde{\boldsymbol{\omega}}(l,\varepsilon) - \frac{1}{\zeta l}[\boldsymbol{I} - \tilde{\boldsymbol{\gamma}}(l,\varepsilon)\boldsymbol{\gamma}_R(\varepsilon)]\,\tilde{\boldsymbol{N}}_R(\varepsilon)\,[\tilde{\boldsymbol{\gamma}}_R(\varepsilon) - \tilde{\boldsymbol{\gamma}}(l,\varepsilon)] = \boldsymbol{0}. \tag{16}$$

The goal of exercises 2a) and 2b) is to rewrite the differential equations given by Eqs. (9) – (12) into a form that is more suitable for numerical calculations. We want to reduce the problem from four 2×2 *complex* equations to one *real* equation with a 32–component vector. The goal of the numerical procedure is to find this 32–component vector $\vec{v}$ satisfying the vectorized Usadel equation,

$$\partial_x \vec{v}(x,\varepsilon) = \vec{f}(x,\varepsilon).$$

However, we still need matrix products in the Usadel equation, so we need functions for transforming back and forth between matrices and vectors.

Exercise 2a) Make a function that transforms a 2×2 *complex* matrix to a *real* vector with eight components. Let $a_r, b_r, c_r, d_r, a_i, b_i, c_i$ and d_i be real numbers. The matrix

$$\boldsymbol{M} = \begin{pmatrix} a_r + \mathrm{i}a_i & b_r + \mathrm{i}b_i \\ c_r + \mathrm{i}c_i & d_r + \mathrm{i}d_i \end{pmatrix} \tag{17}$$

is transformed into the vector

$$\vec{m} = \begin{pmatrix} a_r & b_r & c_r & d_r & a_i & b_i & c_i & d_i \end{pmatrix}. \tag{18}$$

In $\boldsymbol{M}$, i is the imaginary unit. Make the inverse function transforming $\vec{m}$ to $\boldsymbol{M}$ as well. Control that they indeed are inverse and work correctly, and explain how you checked it.

Hint: Use `numpy.real` and `numpy.imag`.

Exercise 2b) Make a function that takes four real 8-component vectors $\vec{m}_1$, $\vec{m}_2$, $\vec{m}_3$, $\vec{m}_4$ as input. The function transforms the vectors into one 32–component real vector $\vec{v}$. Make an inverse function. Control that they work correctly and explain how you checked it.

Hint: Use `numpy.concatenate` and array slicing.

Exercise 2c) You now have the tools for writing the unknown matrices $\boldsymbol{\gamma}(x,\varepsilon)$, $\tilde{\boldsymbol{\gamma}}(x,\varepsilon)$, $\boldsymbol{\omega}(x,\varepsilon)$ and $\tilde{\boldsymbol{\omega}}(x,\varepsilon)$ into one vector $\vec{v}$. Write a function that does this, and also write the inverse function. Show how you transform between $\vec{v}$ and the matrices $\boldsymbol{\gamma}(x,\varepsilon)$, $\tilde{\boldsymbol{\gamma}}(x,\varepsilon)$, $\boldsymbol{\omega}(x,\varepsilon)$ and $\tilde{\boldsymbol{\omega}}(x,\varepsilon)$ in your code. Write down all the 32 components of the vector $\vec{v}$.

Exercise 2d) Make a function that takes $\vec{v}$ and ε as input and returns the derivative $\partial_x \vec{v}(x,\varepsilon)$. This is the Usadel equation for one energy and one position.

Exercise 2e) Make a function that takes two inputs: a vector $\vec{x}$ with m components and a matrix `vec` with dimensions $(32, m)$ (i.e. a $32 \times m$ matrix). The matrix contains the vectors $\vec{v}$ at each position on $\vec{x}$. The function should return a $(32, m)$ matrix containing $\partial_x \vec{v}$ at each position. The function you write now will be the function `fun` in `solve_bvp`. Make sure that your function is on the format that `solve_bvp` requires.

Note: The vector $\vec{x}$ is not needed for calculating $\partial_x \vec{v}$. However, `solve_bvp` requires this vector as an input.

Note: If you write the function in exercise 2e) without any for-loops, but rather use `numpy` functions such as `numpy.einsum`, your code in the coming exercises will run significantly faster. You do *not* need to do this to get a full score, and for the testing phase I do not recommend doing this as it makes debugging harder.

Exercise 2f) Make a function that takes the 32–component vectors $\vec{v}_{\text{left}}$ and $\vec{v}_{\text{right}}$ as input. These contain the Riccati matrices and their derivatives at the left and right edges of the normal metal, respectively. The function calculates the left-hand sides of Eqs. (13)–(16) and returns these values as one real 32–component vector. Assume that the metal is interfaced with two normal metals, for which the Riccati matrices are zero: $\boldsymbol{\gamma}_L(\varepsilon), \tilde{\boldsymbol{\gamma}}_L(\varepsilon), \boldsymbol{\gamma}_R(\varepsilon), \tilde{\boldsymbol{\gamma}}_R(\varepsilon) = \mathbf{0}$. The function you write now will be the function `bc` in `solve_bvp`. Make sure that your function is on the format that `solve_bvp` requires.

Exercise 2g) Use `solve_bvp`, the functions you wrote in exercise 2e) and 2f) as well as the m-component vector $\vec{x} =$ `np.linspace(0,`l`,m)` (the positions) and the initial guess `y=numpy.zeros((32,m))` to solve the Usadel equations. Solve the equations for $\varepsilon \in \{0, 1, 2\}$, $l = 1$, and let $m = 101$. Physically, we have interfaced a normal metal with two normal metals and we expect nothing interesting to happen. In particular, the solution should be identical to a normal metal. Confirm that all components in your $(32, m)$ solution are numerically identical to zero.

Exercise 2h) Use your solution from exercise 2g) and make a plot showing the density of states D/D_0 as a function of position for the three energies. The density of states is given by

$$D(\varepsilon, x)/D_0 = \Re[\mathrm{Tr}(\hat{\rho}_3 \hat{g}(x, \varepsilon)]/4, \tag{19}$$

where Tr means matrix trace, $\Re$ is the real value, and the expression for $\hat{g}(x, \varepsilon)$ is given by eq. (8). D_0 is the normal-state density of states. The matrix $\hat{\rho}_3$ is given by

$$\hat{\rho}_3 = \begin{pmatrix} \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{I} \end{pmatrix}.$$

Exercise 2i) Repeat exercise 2f), but this time assume that the normal metal is interfaced with two superconductors as in Fig. 2. The Riccati matrices for the superconductors at the interfaces are given by

$$\boldsymbol{\gamma}_L(\varepsilon) = \begin{pmatrix} 0 & \frac{s_+(\varepsilon)}{1+c_+(\varepsilon)} \\ \frac{s_-(\varepsilon)}{1+c_-(\varepsilon)} & 0 \end{pmatrix} e^{i\phi_L} \qquad \tilde{\boldsymbol{\gamma}}_L(\varepsilon) = \begin{pmatrix} 0 & \frac{s_-(\varepsilon)}{1+c_-(\varepsilon)} \\ \frac{s_+(\varepsilon)}{1+c_+(\varepsilon)} & 0 \end{pmatrix} e^{-i\phi_L}$$

$$\boldsymbol{\gamma}_R(\varepsilon) = \begin{pmatrix} 0 & \frac{s_+(\varepsilon)}{1+c_+(\varepsilon)} \\ \frac{s_-(\varepsilon)}{1+c_-(\varepsilon)} & 0 \end{pmatrix} e^{i\phi_R} \qquad \tilde{\boldsymbol{\gamma}}_R(\varepsilon) = \begin{pmatrix} 0 & \frac{s_-(\varepsilon)}{1+c_-(\varepsilon)} \\ \frac{s_+(\varepsilon)}{1+c_+(\varepsilon)} & 0 \end{pmatrix} e^{-i\phi_R}$$

where $s_\sigma = \sinh(\vartheta_\sigma)$, $c_\sigma = \cosh(\vartheta_\sigma)$, $\vartheta_\sigma = \mathrm{atanh}(\sigma/(\varepsilon + i\delta))$, and $\sigma = \pm 1$. ϕ_L and ϕ_R are the phases of the left and right superconductors, respectively. Set the phases $\phi_L = \phi_R = 0$.

Exercise 2j) Use `solve_bvp`, the functions you wrote in exercise 2e) and 2i) as well as the m-component vector $\vec{x} =$ `np.linspace(0,`l`,m)` and the initial guess `y=np.zeros((32,m))` to solve the Usadel equation for $\varepsilon = 2$. Plot the density of states as a function of position. What do you expect considering Fig. 1 and the result in exercise 2h)?

In the next exercise, we solve the Usadel equation for energies in the interval $[0, 2]$. At high energies, far away from the superconducting gap, superconductors behave like normal metals. Therefore, `y=np.zeros((32,100))` is a reasonable guess for $\varepsilon = 2$. However, at lower energies this is not a reasonable guess. If the Usadel equation has multiple mathematical solutions, this guess might yield an unphysical solution. Therefore, we solve the Usadel equation first for $\varepsilon = 2$, and then we use this solution as an initial guess for the next energy. This means that even if we only want the solution at for example $\varepsilon = 0$, we have to start at $\varepsilon = 2$ and calculate the solution for all the energies successively until we reach zero.

Exercise 2k) Solve the Usadel equation for 101 energies in the interval $\varepsilon \in [0, 2]$. Plot the density of states in the middle of the normal metal ($x = l/2$) as a function of energy. Do this for $l = 0.5$, $l = 1$ and $l = 2$.

You should expect to see a gap in the density of states in the normal metal. This gap is called a *minigap*, and it shows up because superconducting properties are "leaking" into the normal metal. This is known as the proximity effect. What happens to the minigap when the length increases?

The current through the normal metal at temperature $T = 0$ K (absolute zero) is given by

$$I(x) = -I_0 \int_0^\infty j(x, \varepsilon) d\varepsilon \tag{20}$$

and the current integrand $j(x, \varepsilon) = \Re\{\text{Tr}\left(\hat{\rho}_3 \left[\hat{g}(x, \varepsilon)\frac{d\hat{g}(x,\varepsilon)}{dx} - \frac{d\hat{g}(x,\varepsilon)}{dx}\hat{g}(x, \varepsilon)\right]\right)\}$. Here again, $\Re$ is the real value and Tr is the matrix trace. The derivative of the Green function is given by

$$\partial_x \hat{g}(x, \varepsilon) = 2\begin{pmatrix} \partial_x \boldsymbol{N}(x, \varepsilon) & \boldsymbol{N}(x, \varepsilon)\boldsymbol{\omega}(x, \varepsilon) + [\partial_x \boldsymbol{N}(x, \varepsilon)]\boldsymbol{\gamma}(x, \varepsilon) \\ -\tilde{\boldsymbol{N}}(x, \varepsilon)\tilde{\boldsymbol{\omega}}(x, \varepsilon) - [\partial_x \tilde{\boldsymbol{N}}(x, \varepsilon)]\tilde{\boldsymbol{\gamma}}(x, \varepsilon) & -\partial_x \tilde{\boldsymbol{N}}(x, \varepsilon) \end{pmatrix}. \tag{21}$$

Here, we have

$$\partial_x \boldsymbol{N}(x, \varepsilon) = \boldsymbol{N}(x, \varepsilon)[\boldsymbol{\omega}(x, \varepsilon)\tilde{\boldsymbol{\gamma}}(x, \varepsilon) + \boldsymbol{\gamma}(x, \varepsilon)\tilde{\boldsymbol{\omega}}(x, \varepsilon)]\boldsymbol{N}(x, \varepsilon)$$

and

$$\partial_x \tilde{\boldsymbol{N}}(x, \varepsilon) = \tilde{\boldsymbol{N}}(x, \varepsilon)[\tilde{\boldsymbol{\omega}}(x, \varepsilon)\boldsymbol{\gamma}(x, \varepsilon) + \tilde{\boldsymbol{\gamma}}(x, \varepsilon)\boldsymbol{\omega}(x, \varepsilon)]\tilde{\boldsymbol{N}}(x, \varepsilon).$$

Exercise 2l) Write a function that calculates the current integrand $j(x, \varepsilon)$. Use your solutions from exercise 2k) with $l = 1$ and plot the current integrand as a function of position for $\varepsilon \in \{2, 1.5, 1, 0.5, 0\}$. The current integrand should be zero because there is no phase difference between the superconductors.

Exercise 2m) Set the phases to $\phi_L = 1$ and $\phi_R = 0$. Solve the Usadel equation for the 101 energies in the interval $\varepsilon \in [0, 2]$. Let $l = 1$. Plot the current integrand as a function of energy in the middle of the normal metal ($x = l/2$), and comment on the difference from exercise 2l). Is the current integrand conserved as a function of position?

Note: The current integrand is conserved as a function of position if $j(x, \varepsilon) = j(0, \varepsilon)$ for all x.

Exercise 2n) Solve the Usadel equations for at least 10 evenly spaced phase differences $\Delta\phi = \phi_L - \phi_R \in [0, 2\pi]$. Do this for $\varepsilon \in [0, 2]$ and $l = 1$. Calculate the current integrand, and use Simpson's method (e.g. `scipy.integrate.simpson`) to calculate the current[a]

$$I/I_0 = -\int_0^2 j(x, \varepsilon) d\varepsilon.$$

Plot the current I/I_0 as a function of the phase difference. What do you observe?

[a]Here, we have replaced the upper limit in the integral in eq. (20) with $\varepsilon = 2$. It would be more accurate to use a higher upper bound on the integral ($\varepsilon = 30$ is usually sufficient), and a higher density of energies on the interval $\varepsilon \in [0, 2]$. This is, however, more computationally costly, and since the physics is qualitatively the same we restrict ourselves to an integral over the given energies.