# FEM for Boltzmann Conductivity: An Introduction

Saleh Shamloo Ahmadi

September 15, 2025

## 1 Problem Statement

### 1.1 The Boltzmann Transport Equation

From the Boltzmann transport equation, we have

$$\frac{df}{dt} = \left(\frac{\partial f}{\partial t}\right)_{\text{coll}}, \tag{1}$$

$$\frac{\partial f}{\partial t} + \frac{\mathbf{p}}{m} \cdot \boldsymbol{\nabla}_{\mathbf{r}} f + \mathbf{F} \cdot \boldsymbol{\nabla}_{\mathbf{p}} f = I[f], \tag{2}$$

where $f$ is any density function, $\mathbf{p}$ is the momentum of the particles, $\mathbf{F}$ is the external force, and $I[f]$ is the *collision integral*. In the context of conductivity of electrons,

$$\mathbf{F} = -e(\mathbf{E} + \mathbf{v} \times \mathbf{B}), \tag{3}$$

and replacing the momentum $\mathbf{p}$ with the wavevector $\mathbf{k}$, we have

$$\frac{\partial f}{\partial t} + \frac{\hbar\mathbf{k}}{m} \cdot \boldsymbol{\nabla}_{\mathbf{r}} f - \frac{e}{\hbar}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \boldsymbol{\nabla}_{\mathbf{k}} f = I[f]. \tag{4}$$

If we consider a fluctuation to the equilibrium distribution $f_0(\mathbf{k})$ (the Fermi-Dirac distribution here), then,

$$f(\mathbf{r}, \mathbf{k}; t) = f_0(\mathbf{k}) + g(\mathbf{r}, \mathbf{k}; t) \tag{5}$$

(assuming the material is uniform, the equilibrium distribution does not have any $\mathbf{r}$ dependence). We could also rewrite the wavevector gradient by applying the chain rule and using the relation between the group velocity and the energy. That is,

$$\left.\begin{aligned} \mathbf{v} &= \frac{\boldsymbol{\nabla}_{\mathbf{k}}\varepsilon}{\hbar} \qquad (6)\\ \boldsymbol{\nabla}_{\mathbf{k}} f_0 &= (\boldsymbol{\nabla}_{\mathbf{k}}\varepsilon)\left(\frac{\partial f_0}{\partial \varepsilon}\right), \qquad (7) \end{aligned}\right\} \implies \boldsymbol{\nabla}_{\mathbf{k}} f_0 = \hbar\mathbf{v}\frac{\partial f_0}{\partial \varepsilon}$$

where $\mathbf{v}$ is the group velocity $\varepsilon$ is the energy. Using all this, equation (4) becomes

$$\frac{\partial g}{\partial t} + \frac{\hbar\mathbf{k}}{m} \cdot \boldsymbol{\nabla}_{\mathbf{r}} g - \frac{e(\mathbf{E} + \mathbf{v} \times \mathbf{B})}{\hbar} \cdot \boldsymbol{\nabla}_{\mathbf{k}} g - I[f_0 + g] = e\mathbf{E} \cdot \mathbf{v}\frac{\partial f_0}{\partial \varepsilon} \tag{8}$$

(note that $\mathbf{v} \times \mathbf{B}$ is perpendicular to $\mathbf{v}$). Now, we only consider the case of uniform fields (i.e. no $\mathbf{r}$ dependence), so we can drop the spatial gradient term. Also, for the

conductivity tensor, we are only interested in the linear response to the electric field $\mathbf{E}$, so we can drop the electric field term on the left-hand side ($g$ itself is in first order with respect to the electric field, so multiplying by the electric field itself gives a term of second order in the electric field). Finally, linearizing the collision integral in terms of the fluctuation $g$, we have

$$I[f_0 + g] = \int d\mathbf{k}'\, C(\mathbf{k}, \mathbf{k}')g(\mathbf{k}'; t) - \int d\mathbf{k}'\, C(\mathbf{k}', \mathbf{k})g(\mathbf{k}; t) + \mathcal{O}(g^2), \tag{9}$$

where $C(\mathbf{k}, \mathbf{k}')$ is a scattering kernel. I call the two terms the *in-scattering* and *out-scattering* terms respectively. The out-scattering is usually expressed with the *relaxation time* $\tau$ or *scattering rate* $\Gamma = 1/\tau$, which is equal to

$$\frac{1}{\tau(\mathbf{k})} = \int d\mathbf{k}'\, C(\mathbf{k}', \mathbf{k}). \tag{10}$$

In-scattering is dropped in the *relaxation time approximation*, which is the case for the Chambers formula. Our goal here is to keep this term, as it especially becomes important when there is strong anisotropy in the scattering, which is the case for cuprates. When the scattering is roughly isotropic, $C(\mathbf{k}, \mathbf{k}')$ does not strongly depend on $\mathbf{k}$ and, because of charge conservation, $\int d\mathbf{k}'\, C(\mathbf{k}, \mathbf{k}')g(\mathbf{k}'; t) \approx C \int d\mathbf{k}'\, g(\mathbf{k}'; t) = 0$. Due to time-reversal symmetry, $C(\mathbf{k}, \mathbf{k}') = C(\mathbf{k}', \mathbf{k})$, so if we consider any anisotropy in the out-scattering, we must do the same for the in-scattering. This means the Chambers formula is just wrong for anisotropic scattering. So, why does it work effectively anyway? The scattering in cuprates might only be low-angle, so the in-scattering is small compared to the out-scattering.

Plugging in the expression for the collision integral into equation (8) and applying the approximations, we get

$$\left(\frac{\partial}{\partial t} + \frac{1}{\tau(\mathbf{k})} - \frac{e(\mathbf{v} \times \mathbf{B})}{\hbar} \cdot \boldsymbol{\nabla}_{\mathbf{k}}\right) g(\mathbf{k}; t) - \int d\mathbf{k}'\, C(\mathbf{k}, \mathbf{k}')g(\mathbf{k}'; t) = e\mathbf{E} \cdot \mathbf{v} \frac{\partial f_0}{\partial \varepsilon}. \tag{11}$$

Depending on whether we want to study steady-state DC fields or AC fields, we can set $\partial/\partial t$ to equal to zero or perform a Fourier transform and set $\partial/\partial t = -i\omega$. For the sake of completeness, I will go with the latter case (just setting $\omega = 0$ and using the regular, non-Fourier-trasformed fields would produce the former case easily).

This equation can be expressed as

$$\int d\mathbf{k}'\, v(\mathbf{k}) A(\mathbf{k}, \mathbf{k}') g(\mathbf{k}') = h(\mathbf{k}) \tag{12}$$

where

$$A(\mathbf{k}, \mathbf{k}') = \left(-\frac{i\omega}{v(\mathbf{k})} + \frac{1}{v(\mathbf{k})\tau(\mathbf{k})} - \frac{e(\hat{\mathbf{v}}(\mathbf{k}) \times \mathbf{B})}{\hbar} \cdot \boldsymbol{\nabla}_{\mathbf{k}}\right) \delta(\mathbf{k} - \mathbf{k}') - \frac{C(\mathbf{k}, \mathbf{k}')}{v(\mathbf{k})}, \tag{13}$$

$$h(\mathbf{k}) = e\mathbf{E} \cdot \mathbf{v}(\mathbf{k}) \frac{\partial f_0}{\partial \varepsilon}. \tag{14}$$

Where $v = \|\mathbf{v}\|$ and $\hat{\mathbf{v}} = \mathbf{v}/v$. Separating the velocity magnitude $v(\mathbf{k})$ seems like an arbitrary choice here, but it is done with good reason. Later, it will make the formula

for the conductivity tensor more symmetric and makes the calculation of $A$ a bit more streamlined. The solution can be expressed as

$$g(\mathbf{k}) = \frac{1}{v(\mathbf{k})} \int d\mathbf{k}' \, A^{-1}(\mathbf{k}, \mathbf{k}') h(\mathbf{k}'). \tag{15}$$

So, if we could discretize everything, this would just become a linear system of equations. This is the goal of the finite element method (FEM).

### 1.2 The Conductivity Tensor

The current density is given by

$$\mathbf{J} = -\frac{2e}{(2\pi)^d} \int_{\mathrm{BZ}} d\mathbf{k} \, \mathbf{v}(\mathbf{k}) f(\mathbf{k}), \tag{16}$$

where BZ is the Brillouin zone. The factor of 2 in the numerator is due to the two possible spin configurations per wavevector $k$ and the factor $(2\pi)^d$ in the denominator comes from the weighted sum over the wavevectors. The conductivity tensor is given by

$$\sigma_{ab} = \frac{\partial J_a}{\partial E_b}. \tag{17}$$

Since there are no currents in equilibrium, we have

$$\sigma_{ab} = -\frac{2e}{(2\pi)^d} \int_{\mathrm{BZ}} d\mathbf{k} \, v_a(\mathbf{k}) \frac{\partial g(\mathbf{k})}{\partial E_b}. \tag{18}$$

plugging in the expression for $g(\mathbf{k})$,

$$\sigma_{ab} = -\frac{2e^2}{(2\pi)^d} \int_{\mathrm{BZ}} d\mathbf{k} \, \frac{v_a(\mathbf{k})}{v(\mathbf{k})} \int d\mathbf{k}' \, A^{-1}(\mathbf{k}, \mathbf{k}') v_b(\mathbf{k}') \left( \frac{\partial f_0}{\partial \varepsilon} \right). \tag{19}$$

As $T \to 0$, the Fermi-Dirac distribution becomes a step function, so the term $\partial f_0 / \partial \varepsilon$ becomes a Dirac delta at the Fermi energy. If we apply the integral over the delta function, an extra factor of $(\boldsymbol{\nabla}_k \varepsilon)^{-1} = (\hbar v)^{-1}$ (the Jacobian) is added. In the end, we have

$$\sigma_{ab} = -\frac{2e^2}{(2\pi)^d \hbar} \int_{\mathrm{FS}} d\mathbf{k} \int d\mathbf{k}' \, \hat{v}_a(\mathbf{k}) A^{-1}(\mathbf{k}, \mathbf{k}') \hat{v}_b(\mathbf{k}'), \tag{20}$$

where FS is the Fermi surface, and $\hat{v}_a = v_a / v$ is the component of the velocity in the $a$ direction, divided by the full magnitude of the velocity.

## 2 Finite Element Method

### 2.1 Discretization

To solve this using the finite element method, we first define a set of basis functions, and then approximate everything as a linear combination of these basis functions. So, the goal is to express everything in equation (20) in terms of linear combinations of these basis functions and transform the differential equation problem into a linear algebra problem.

To do this, we use the machinery of tensor calculus, but it is not necessary to have a deep understanding of it. It is only used to simplify the notation.

Before we begin, remember the simple rule that when summing over an index, it should be on the top for one quantity and on the bottom for another. If an index is not summed over, then it will obviously stay where it is in the final quantity. We call this the *index contraction* rule. The placement of indices on the top or the bottom has a deeper connection with tensor calculus and covariance and contravariance, but it is not needed to understand the procedures here. You can just think of it as a syntactic game. I will also not use the Einstein summation convention, just to make things completely explicit and hopefully clearer.

Defining the basis as the set $\{\psi_i(\mathbf{k})\}$, we can approximate any function by a linear combination of these basis functions;

$$f(\mathbf{k}) = \sum_i f^i \psi_i(\mathbf{k}). \tag{21}$$

Now, we would like to evaluate integrals over the Fermi surface. So, to make our notation simpler, we can define quantities with subscripts to represent integrals. Also, notice these are projections of the functions onto the basis functions.

$$f_i \equiv \int d\mathbf{k}\, \psi_i(\mathbf{k}) f(\mathbf{k}) = \sum_j f^j \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}) \tag{22}$$

It is useful to define the quantity

$$M_{ij} \equiv \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}), \tag{23}$$

which we call the *overlap matrix*. Using it,

$$f_i = \sum_j M_{ij} f^j. \tag{24}$$

We now try to define the operator in this basis. The question is, where should we put the indices for the matrix representing the action of operator $A$ in this space? Well, the action of such matrix on a vector $f^i$ should produce a new vector $(Af)^i$. This should be a linear combination of the components of the vector $f^i$. So, to abide by our rule of index contraction, the symbol for the matrix of $A$ should have one index on the bottom to sum over the elements of $f^i$ and one index on the top for each of the elements of the resulting vector $(Af)^i$. Putting it all together,

$$(Af)(\mathbf{k}) = \sum_i (Af)^i \psi_i(\mathbf{k}) \equiv \sum_{ij} A^i_j f^j \psi_i(\mathbf{k}). \tag{25}$$

For the integrals, we have

$$(Af)_i = \int d\mathbf{k}\, \psi_i(\mathbf{k})(Af)(\mathbf{k}) = \sum_{jk} A^j_k f^k \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}) = \sum_{jk} M_{ij} A^j_k f^k. \tag{26}$$

To make it simpler, we can define the symbol

$$A_{ij} \equiv \sum_k M_{ik} A^k_j \tag{27}$$

so that

$$(Af)_i = \sum_j A_{ij} f^j. \tag{28}$$

From equations (24) and (27), we can see that the overlap matrix $M$ performs the action of *lowering* the indices. If we invert this matrix in each of these linear systems, we have

$$f^i = \sum_j (M^{-1})^{ij} f_j, \tag{29}$$

$$A^i_j = \sum_k (M^{-1})^{ik} A_{kj}. \tag{30}$$

Again, the index placement is done by the index contraction rule. So we can see that the inverse of the overlap matrix $M^{-1}$ performs the action of *raising* the indices. In tensor algebra, the equivalent of this is the metric tensor.

Finding the components of the vectors $f^i$ is easy; they are just the value of the functions at each discrete point in our approximation space. But how do we find the elements of the matrix $A$? The action of the operator $A$ on a function $f(\mathbf{k})$ is

$$(Af)(\mathbf{k}) = \int d\mathbf{k}'\, A(\mathbf{k}, \mathbf{k}') f(\mathbf{k}'). \tag{31}$$

Inserting this into equation (26), and utilizing equation (28), we get

$$\sum_j A_{ij} f^j = \int d\mathbf{k}\, \psi_i(\mathbf{k}) \int d\mathbf{k}'\, A(\mathbf{k}, \mathbf{k}') f(\mathbf{k}'). \tag{32}$$

expanding the function $f(\mathbf{k}')$ in terms of the basis functions from equation (21),

$$\sum_j A_{ij} f^j = \int d\mathbf{k}\, \psi_i(\mathbf{k}) \int d\mathbf{k}'\, A(\mathbf{k}, \mathbf{k}') \sum_j f^j \psi_j(\mathbf{k}'). \tag{33}$$

This is true for any function vector $f^j$, so we can drop it from both sides. This finally gives

$$A_{ij} = \int d\mathbf{k} \int d\mathbf{k}'\, \psi_i(\mathbf{k}) A(\mathbf{k}, \mathbf{k}') \psi_j(\mathbf{k}'). \tag{34}$$

Now that we know how to expand everything in this discrete basis, let us go back to our original integral equation (12). We would like to solve

$$\int d\mathbf{k}'\, A(\mathbf{k}, \mathbf{k}') g(\mathbf{k}') = (Ag)(\mathbf{k}) = h(\mathbf{k}). \tag{35}$$

To express this equation in terms of the symbols we know, we can multiply both sides by $\psi_i(\mathbf{k})$ and integrate over $\mathbf{k}$. This gives

$$\int d\mathbf{k}\, \psi_i(\mathbf{k}) (Ag)(\mathbf{k}) = \int d\mathbf{k}\, \psi_i(\mathbf{k}) h(\mathbf{k}). \tag{36}$$

Using equations (22) and (28), this becomes

$$(Ag)_i = h_i. \tag{37}$$

Expanding the left-hand side,

$$\sum_j A_{ij} g^j = h_i \tag{38}$$

and solving the linear system,

$$g^i = \sum_j (A^{-1})^{ij} h_j. \tag{39}$$

Again, the index placement is done by the index contraction rule. The important result is that we need to invert the $A_{ij}$ form of the matrix to get the inverse we need for our case.

$$(A^{-1})^{ij} = (A_{ij})^{-1}. \tag{40}$$

Finally, to calculate the conductivity tensor from (20), we need to multiply $\hat{v}_a$ with $A^{-1}\hat{v}_b$ and take an integral over the result. To express this operation in our discrete basis, consider two functions $f$ and $g$. Their product is

$$f(\mathbf{k})g(\mathbf{k}) = \sum_{ij} f^i g^j \psi_i(\mathbf{k})\psi_j(\mathbf{k}), \tag{41}$$

and the integral over this product is

$$\int d\mathbf{k}\, f(\mathbf{k})g(\mathbf{k}) = \sum_{ij} f^i g^j \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}) = \sum_{ij} f^i g^j M_{ij} = \sum_i f^i g_i. \tag{42}$$

So, the vector for one function needs to have indices on top and the other on the bottom. From equation (39), we can see $A^{-1}\hat{v}_b$ has the form $(A^{-1})^{ij}(\hat{v}_b)_j$, which would have an index on top after contraction. Therefore, $\hat{v}_a$ would have to have an index on the bottom to perform the multiplication and integration.

Putting everything together, we have

$$\sigma_{ab} = -\frac{2e^2}{(2\pi)^d \hbar} \sum_{ij} (\hat{v}_a)_i \left(A^{-1}\right)^{ij} (\hat{v}_b)_j, \tag{43}$$

or, in terms of all the terms we know how to calculate,

$$\sigma_{ab} = -\frac{2e^2}{(2\pi)^d \hbar} \sum_{ij} \left[ \left( \sum_k M_{ik} (\hat{v}_a)^k \right) (A_{ij})^{-1} \left( \sum_k M_{jk} (\hat{v}_b)^k \right) \right]. \tag{44}$$

To do the computation in practice, $M_{ij}$ and $A_{ij}$ can be calculated from (23) and (34) respectively, and $(\hat{v}_a)^i$ and $(\hat{v}_b)^i$ vector components are the values of $\hat{v}_a$ and $\hat{v}_b$ at the discrete points of our approximation space. Afterwards, all that will be left to do is two matrix multiplications $M\hat{v}$, one linear system solution $A^{-1}(M\hat{v}_b)$, and one dot product $(M\hat{v}_a) \cdot (A^{-1}M\hat{v}_b)$.

## 2.2 Differential Operator Terms

Splitting $A$ into the comprising terms using (13) and (34), we have

$$\begin{aligned} A_{ij} = & \int d\mathbf{k} \left( -i\omega + \frac{1}{\tau(\mathbf{k})} \right) \frac{\psi_i(\mathbf{k})\psi_j(\mathbf{k})}{v(\mathbf{k})} \\ & - \int d\mathbf{k}\, \psi_i(\mathbf{k}) \frac{e(\hat{\mathbf{v}}(\mathbf{k}) \times \mathbf{B})}{\hbar} \cdot \boldsymbol{\nabla}_{\mathbf{k}} \psi_j(\mathbf{k}) \\ & - \int d\mathbf{k} \int d\mathbf{k}'\, \psi_i(\mathbf{k}) \frac{C(\mathbf{k}, \mathbf{k}')}{v(\mathbf{k})} \psi_j(\mathbf{k}') \end{aligned} \tag{45}$$

$$\equiv \Gamma_{ij} - \frac{eB}{\hbar} \left( D_{\hat{\mathbf{v}} \times \hat{\mathbf{B}}} \right)_{ij} - S_{ij}. \tag{46}$$

The matrices $\Gamma_{ij}$, $D_{\hat{\mathbf{v}} \times \hat{\mathbf{B}}}$ and $S_{ij}$ are called the *out-scattering*, *derivative* and *in-scattering* matrices respectively.

Expanding the $k$-dependence in terms of the basis functions for the out-scattering matrix $\Gamma_{ij}$ gives

$$\Gamma_{ij} = \sum_k \int d\mathbf{k} \left( -i\omega + \frac{1}{\tau_k} \right) \frac{\psi_i(\mathbf{k})\psi_j(\mathbf{k})\psi_k(\mathbf{k})}{v_k}. \tag{47}$$

Let us define the tensor $\Pi_{ijk}$ and the vector $\gamma^k$, such that

$$\Gamma_{ij} = \textstyle\sum_k \Pi_{ijk} \gamma^k \implies \begin{cases} \Pi_{ijk} = \displaystyle\int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k})\psi_k(\mathbf{k}), & (48) \\ \gamma^k = -\dfrac{i\omega}{v_k} + \dfrac{1}{v_k \tau_k}. & (49) \end{cases}$$

So, for any given basis, we have to calculate the $\Pi_{ijk}$ tensor. The $\gamma^k$ vector is given by the velocity, frequency, and scattering kernel (or possibly the scattering rate $1/\tau$ defined "by hand" when ignoring in-scattering).

Expanding the $k$-dependence in terms of the basis functions for the in-scattering matrix $S_{ij}$ gives

$$S_{ij} = \sum_{mn} \int d\mathbf{k} \int d\mathbf{k}'\, \psi_i(\mathbf{k})\psi_m(\mathbf{k}) \frac{C_{mn}}{v_m} \psi_n(\mathbf{k}')\psi_j(\mathbf{k}'). \tag{50}$$

Let us define the tensor $\Xi_{ijmn}$ and the matrix $s^{mn}$, such that

$$S_{ij} = \textstyle\sum_{m,n} \Xi_{ijmn} s^{mn} \implies \begin{cases} \Xi_{ijmn} = \displaystyle\int d\mathbf{k} \int d\mathbf{k}'\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}')\psi_m(\mathbf{k})\psi_n(\mathbf{k}'), & (51) \\ s^{mn} = \dfrac{C_{mn}}{v_m}. & (52) \end{cases}$$

So, for any given basis, we have to calculate the $\Xi_{ijmn}$ tensor and the $s^{mn}$ matrix is given from the scattering kernel.

## 2.3 Piecewise Linear Basis for 2D Fermi Surfaces

The 2D case is a bit simpler and can help you gain a better understanding before moving to the full 3D case.

We choose $n$ points on $\mathbf{k}_i$ to represent the Fermi surface. Then, the discretization of the Fermi surface would be the line segments $I_i$ connecting these points. These line segments are also called the *elements*. The length of each line segment is

$$\ell_i = |\mathbf{k}_{i+1} - \mathbf{k}_i|. \tag{53}$$

Note that since our domain is a closed curve (the Fermi surface is closed), there are periodic boundary conditions, which means the point $i = n + 1$ is the same as $i = 1$ and $i = 0$ is the same as $i = n$. The simplest continuous basis one can define on these elements is the piecewise linear basis. The basis function for each point $\mathbf{k}_i$ on the Fermi surface is linear on the connected elements and peaks at the point $\mathbf{k}_i$, equal to 1. It is zero everywhere else. So, with the parametrization of each line segment $I_i$ as

$$x : I_i \mapsto [0, 1], \tag{54}$$

then the basis functions can be defined as

$$\psi_i(\mathbf{k}) = \begin{cases} x, & \mathbf{k} \in I_{i-1}; \quad (55) \\ 1 - x, & \mathbf{k} \in I_i; \quad (56) \\ 0, & \mathbf{k} \in I_j,\ j \notin \{i-1, i\}. \quad (57) \end{cases}$$

We can now calculate the matrices $M_{ij}$ and $A_{ij}$ in this basis. It is easy to see that, because each element in these matrices contains the multiplication of two basis functions, they are only nonzero when $i$ and $j$ are associated with the same point or neighboring points.

For the overlap matrix,

$$\begin{aligned} M_{ii} &= \int dk\, \psi_i(\mathbf{k})\psi_i(\mathbf{k}) = \int_{I_{i-1}} d\mathbf{k}\, [\psi_i(\mathbf{k})]^2 + \int_{I_i} d\mathbf{k}\, [\psi_i(\mathbf{k})]^2 \\ &= \int_0^1 dx\, x^2 \ell_{i-1} + \int_0^1 dx\, (1-x)^2 \ell_i = \frac{\ell_{i-1} + \ell_i}{3}, \end{aligned} \tag{58}$$

$$M_{i,i+1} = M_{i+1,i} = \int_{I_i} d\mathbf{k}\, \psi_i(\mathbf{k})\psi_{i+1}(\mathbf{k}) = \int_0^1 dx\, x(1-x)\ell_i = \frac{\ell_i}{6}. \tag{59}$$

Therefore, the full overlap matrix is

$$M_{ij} = \frac{\ell_{i-1} + \ell_i}{3}\delta_{ij} + \frac{\ell_i}{6}(\delta_{i+1,j} + \delta_{i,j+1}). \tag{60}$$

For the derivative term $D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}}$, note that the part

$$(\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}) \cdot \boldsymbol{\nabla}_{\mathbf{k}} \tag{61}$$

is the directional derivative in the direction $\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}$. Since the functions are only defined on the discretized 2D rings in $k$-space, this is the dot product of the vector $\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}$ with the direction of the vector $\mathbf{k}_{i+1} - \mathbf{k}_i$ (which is along the line segment $I_i$), which we call $\Delta\hat{\mathbf{k}}_i$. The diagonal terms vanish, since the integral over $I_i$ and $I_{i-1}$ are completely

the same, but with opposite signs; in the direction of $\mathbf{k}_{i-1}$ to $\mathbf{k}_i$, the function $\psi_i$ increases on $I_{i-1}$, and in the direction of $\mathbf{k}_i$ to $\mathbf{k}_{i+1}$, it decreases on $I_i$. The neighboring terms are

$$\begin{aligned}\left(D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}}\right)_{i,i+1} &= -\left(D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}}\right)_{i+1,i} = \int_{I_i} d\mathbf{k}\,\psi_i(\mathbf{k})\frac{\left(\hat{\mathbf{v}}_i\times\hat{\mathbf{B}}\right)\cdot\Delta\hat{\mathbf{k}}_i}{\ell_i} \\ &= \int_0^1 dx\,(1-x)\Big[\left(\hat{\mathbf{v}}_i\times\hat{\mathbf{B}}\right)\cdot\Delta\hat{\mathbf{k}}_i\Big] = \frac{\left(\hat{\mathbf{v}}_i\times\hat{\mathbf{B}}\right)\cdot\Delta\hat{\mathbf{k}}_i}{2},\end{aligned} \tag{62}$$

So, the full matrix is

$$\left(D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}}\right)_{ij} = \left(\hat{\mathbf{v}}_i\times\hat{\mathbf{B}}\right)\cdot\Delta\hat{\mathbf{k}}_i\left(\frac{\delta_{i,j+1}-\delta_{i+1,j}}{2}\right). \tag{63}$$

For the out-scattering term $\Gamma_{ij} = \sum_k \Pi_{ijk}\gamma^k$, only the elements where $i$, $j$ and $k$ are all associated with the same point or neighboring points are nonzero.

$$\Pi_{iii} = \int_{I_i} d\mathbf{k}\,[\psi_i(\mathbf{k})]^3 + \int_{I_{i-1}} d\mathbf{k}\,[\psi_i(\mathbf{k})]^3 = \int_0^1 dx\,x^3\ell_i + \int_0^1 dx\,(1-x)^3\ell_{i-1} = \frac{\ell_i+\ell_{i-1}}{4}, \tag{64}$$

$$\Pi_{i+1,i,i} = \Pi_{i,i+1,i} = \Pi_{i,i,i+1} = \int_{I_i} d\mathbf{k}\,[\psi_i(\mathbf{k})]^2\psi_{i+1}(\mathbf{k}) = \int_0^1 dx\,(1-x)^2x\ell_i = \frac{\ell_i}{12}, \tag{65}$$

$$\Pi_{i,i+1,i+1} = \Pi_{i+1,i,i+1} = \Pi_{i+1,i+1,i} = \int_{I_i} d\mathbf{k}\,\psi_i(\mathbf{k})[\psi_{i+1}(\mathbf{k})]^2 = \int_0^1 dx\,x(1-x)^2\ell_i = \frac{\ell_i}{12}, \tag{66}$$

$$\Pi_{ijk} = \frac{\ell_{i-1}+\ell_i}{4}\delta_{ijk} + \frac{\ell_i}{12}(\delta_{i+1,j,k}+\delta_{i,j-1,k}) + \frac{\ell_j}{12}(\delta_{i,j+1,k}+\delta_{i,j,k-1}) + \frac{\ell_k}{12}(\delta_{i,j,k+1}+\delta_{i-1,j,k}) \tag{67}$$

Multiplying this tensor by $\gamma^k$, we get the full $\Gamma_{ij}$ matrix.

$$\begin{aligned}\Gamma_{ij} = \left(\frac{\ell_{i-1}+\ell_i}{4}\gamma^i + \frac{\ell_i}{12}\gamma^{i+1} + \frac{\ell_{i-1}}{12}\gamma^{i-1}\right)\delta_{ij} + \frac{\ell_i}{12}\left(\gamma^i+\gamma^{i+1}\right)\delta_{i+1,j} \\ + \frac{\ell_j}{12}\left(\gamma^j+\gamma^{j+1}\right)\delta_{i,j+1}\end{aligned} \tag{68}$$

## 2.4 Piecewise Linear Basis for 3D Fermi Surfaces

The discretization procedure for the 3D case can be a bit more complicated. In addition to points, we also need to define some connecting surface for sets of points. In 2D, this was easy because we could just connect pairs of points with line segments. In 3D, to make the elements as simple as possible, we use triangles.

We define the Fermi surface by a set of points $\mathbf{k}_i$ and triangles $T(i,j,k)$ connecting the points $\mathbf{k}_i$, $\mathbf{k}_j$ and $\mathbf{k}_k$. Also, we define $T_i$ as the set of triangles containing the point $\mathbf{k}_i$. Then, defining piecewise linear basis functions, the basis function $\psi_i$ peaks with value 1 at the point $\mathbf{k}_i$, is linear on $T_i$, and is 0 everywhere else.

Each triangle is parametrized as

$$(x,y) : T(i,j,k) \mapsto \{(x,y);\ x\in[0,1],\ y\in[0,1-x]\}, \tag{69}$$

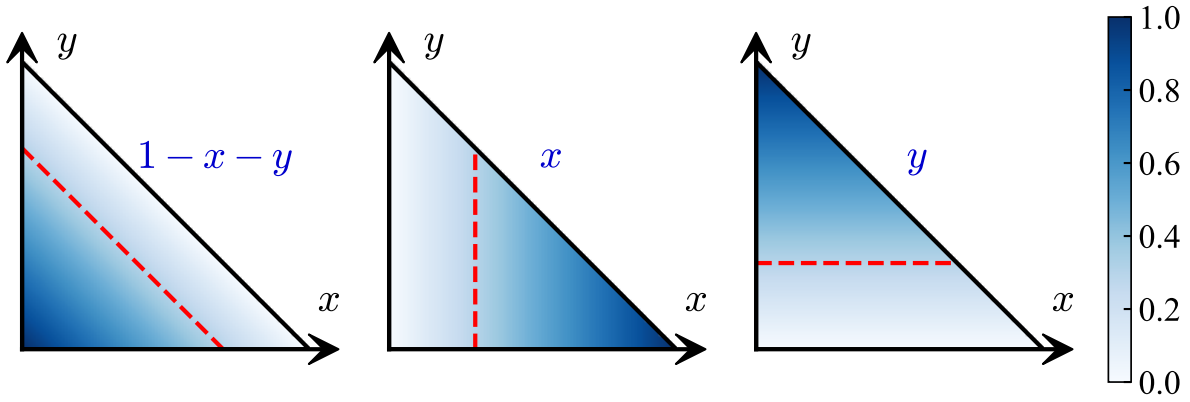


Figure 1: visualization of the basis functions when the function peaks at each of the three vertices of the triangle.

so that each triangle is mapped to a unit right-angled triangle. With this mapping, the basis functions can be defined as

$$\psi_i(\mathbf{k}) = \begin{cases} 1-x-y, & \mathbf{k} \in T_i, (x,y)_{\max(\psi_i)} = (0,0); \quad (70) \\ x, & \mathbf{k} \in T_i, (x,y)_{\max(\psi_i)} = (1,0); \quad (71) \\ y, & \mathbf{k} \in T_i, (x,y)_{\max(\psi_i)} = (0,1); \quad (72) \\ 0, & \mathbf{k} \notin T_i. \quad (73) \end{cases}$$

To visualize these basis functions more easily, you can look at Figure 1.

To compute integrals with this parametrization, we can just use the Jacobian of the transformation. Here, the Jacobian is the ratio of the area of the triangle $T(i,j,k)$ to the area of the unit right-angled triangle. Let us define it as

$$\alpha(i,j,k) = \|(\mathbf{k}_j - \mathbf{k}_i) \times (\mathbf{k}_k - \mathbf{k}_i)\|. \quad (74)$$

Then, the integrals are transformed as

$$\int_{T(i,j,k)} d\mathbf{k} = \alpha(i,j,k) \int_0^1 dx \int_0^{1-x} dy \equiv \alpha(i,j,k) \int_T dx\, dy\,. \quad (75)$$

Now, for calculating each term, note that only the elements where the indices are the same or belong to the same triangle have nonzero overlap (see Figure 2). Here are all the

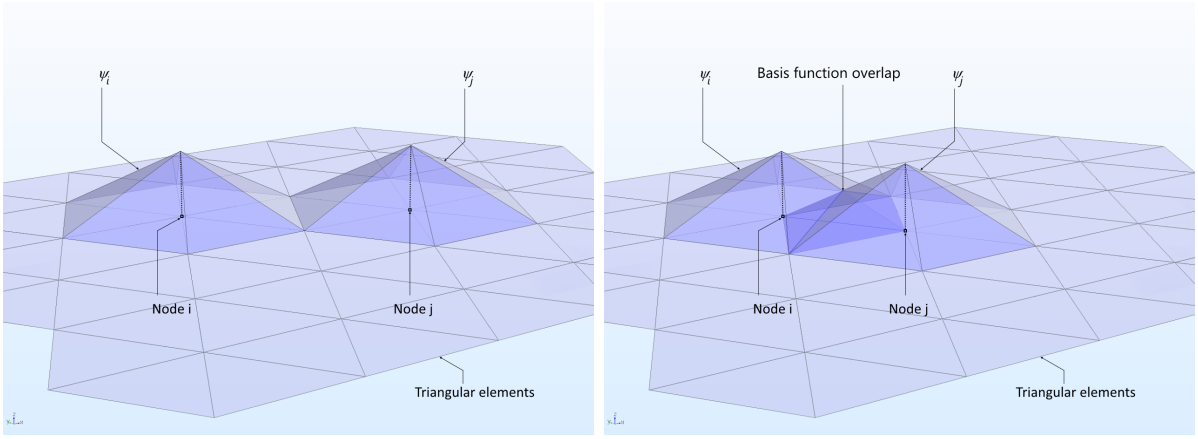


Figure 2: Only neighboring points have overlapping basis functions. Image taken from the COMSOL website.

nonzero terms in the matrices:

$$M_{ii} = \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_i(\mathbf{k}) = \sum_{(i,j,k)\in T_i} \int_{T(i,j,k)} d\mathbf{k}\, (\psi_i(\mathbf{k}))^2 \tag{76}$$

$$= \sum_{(i,j,k)\in T_i} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)^2 = \frac{1}{12} \sum_{(i,j,k)\in T_i} \alpha(i,j,k),$$

$$M_{ij} = \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}) = \sum_{(i,j,k)\in T_i\cap T_j} \int_{T(i,j,k)} d\mathbf{k}\, \psi_i(\mathbf{k})\psi_j(\mathbf{k}) \tag{77}$$

$$= \sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)x = \frac{1}{24} \sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k)$$

$$\left( = \sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)y \right)$$

$$\Pi_{iii} = \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_i(\mathbf{k})\psi_i(\mathbf{k}) = \sum_{(i,j,k)\in T_i} \int_{T(i,j,k)} d\mathbf{k}\, (\psi_i(\mathbf{k}))^3 \tag{78}$$

$$= \sum_{(i,j,k)\in T_i} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)^3 = \frac{1}{20} \sum_{(i,j,k)\in T_i} \alpha(i,j,k),$$

$$\Pi_{iij} = \Pi_{iji} = \Pi_{jii} = \int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_i(\mathbf{k})\psi_j(\mathbf{k}) = \sum_{(i,j,k)\in T_i\cap T_j} \int_{T(i,j,k)} d\mathbf{k}\, (\psi_i(\mathbf{k}))^2\psi_j(\mathbf{k}) \tag{79}$$

$$= \sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)^2 x = \frac{1}{60} \sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k),$$

$$\Pi_{ijk} = \int_{T(i,j,k)} d\mathbf{k}\ \psi_i(\mathbf{k})\psi_j(\mathbf{k})\psi_k(\mathbf{k}) = \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)xy = \frac{1}{120}\alpha(i,j,k). \tag{80}$$

For the derivative term, first, we approximate $\hat{\mathbf{v}}(\mathbf{k})$ as a piecewise constant unit vector field normal to each triangle. That is,

$$\hat{\mathbf{v}}(\mathbf{k}) = \mathrm{sgn}(i,j,k)\hat{\mathbf{n}}(i,j,k); \quad \mathbf{n}(i,j,k) \equiv (\mathbf{k}_j - \mathbf{k}_i) \times (\mathbf{k}_k - \mathbf{k}_i), \tag{81}$$

where $\hat{\mathbf{n}} = \mathbf{n}/\|\mathbf{n}\|$ and $\mathrm{sgn}(i,j,k) = \pm 1$ is chosen such that the normals always point outwards from the surface. Also note that $\|\mathbf{n}(i,j,k)\| = \alpha(i,j,k)$. The other term we need is gradient of $\psi_j$ on triangle $T(i,j,k)$. Since $\psi_j$ is linear on this triangle and peaks at $j$, its gradient is a constant vector pointing to $j$ and perpendicular to the opposite edge $\mathbf{k}_k - \mathbf{k}_i$.

$$\boldsymbol{\nabla}_{\mathbf{k}} \psi_j(\mathbf{k}) = \frac{\mathbf{u}_j(i,j,k)}{u_j^2(i,j,k)}; \tag{82}$$

where $u_j(i,j,k) = \|\mathbf{u}_j(i,j,k)\|$ and

$$\mathbf{u}_j(i,j,k) = (\mathbf{k}_j - \mathbf{k}_i) - \frac{(\mathbf{k}_j - \mathbf{k}_i)\cdot(\mathbf{k}_k - \mathbf{k}_i)}{|\mathbf{k}_k - \mathbf{k}_i|^2}(\mathbf{k}_k - \mathbf{k}_i). \tag{83}$$

The vector $\mathbf{u}_j(i,j,k)$ is obtained by projecting the vector $\mathbf{k}_j - \mathbf{k}_i$ onto the direction perpendicular to the edge $\mathbf{k}_k - \mathbf{k}_i$. This is done by subtracting the component in the direction of $\mathbf{k}_k - \mathbf{k}_i$ from $\mathbf{k}_j - \mathbf{k}_i$. Using these calculations, we can finally compute the derivative term.

$$\left(\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}\right)\cdot \boldsymbol{\nabla}_{\mathbf{k}} \psi_j = (\boldsymbol{\nabla}_{\mathbf{k}} \psi_j \times \hat{\mathbf{v}}(\mathbf{k}))\cdot \hat{\mathbf{B}} = \mathrm{sgn}(i,j,k)\left(\frac{\mathbf{u}_j(i,j,k) \times \hat{\mathbf{n}}(i,j,k)}{u_j^2(i,j,k)}\right)\cdot \hat{\mathbf{B}}. \tag{84}$$

You can simplify this expression by hand. But, using a bit of geometric thinking, it can be done in a simpler way: since $\mathbf{u}_j$ is perpendicular to $\mathbf{k}_i - \mathbf{k}_k$ and $\hat{\mathbf{n}}$ is perpendicular to the plane of the triangle such that it is generated by counterclockwise ("right-handed") cross products of vectors on the plane, $\mathbf{u}_j \times \hat{\mathbf{n}}$ must be in the direction of $\mathbf{k}_i - \mathbf{k}_k$. This means

$$\mathbf{u}_j(i,j,k) \times \hat{\mathbf{n}}(i,j,k) = u_j(i,j,k)\frac{\mathbf{k}_i - \mathbf{k}_k}{\|\mathbf{k}_i - \mathbf{k}_k\|}. \tag{85}$$

Furthermore, $u_j(i,j,k)$ and $\mathbf{k}_i - \mathbf{k}_k$ can be taken as the high and the base of the triangle $T(i,j,k)$ respectively, so the product of their norms is twice the area of the triangle, which is the same as the Jacobian $\alpha(i,j,k)$. Putting it all together, we have

$$\left(\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}\right) \cdot \boldsymbol{\nabla}_{\mathbf{k}} \psi_j = \frac{\mathrm{sgn}(i,j,k)}{\alpha(i,j,k)}(\mathbf{k}_i - \mathbf{k}_k)\cdot \hat{\mathbf{B}}. \tag{86}$$

Using this, we can compute the derivative matrix.

$$\begin{aligned}
(D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}})_{ij} &= \int d\mathbf{k}\, \psi_i(\mathbf{k})\left(\hat{\mathbf{v}}(\mathbf{k}) \times \hat{\mathbf{B}}\right) \cdot \boldsymbol{\nabla}_{\mathbf{k}} \psi_j(\mathbf{k}) \qquad (87)\\
&= \sum_{(i,j,k)\in T_i \cap T_j} \alpha(i,j,k) \int_T dx\, dy\, (1-x-y)\frac{\mathrm{sgn}(i,j,k)}{\alpha(i,j,k)}(\mathbf{k}_i - \mathbf{k}_k)\cdot \hat{\mathbf{B}}\\
&= \frac{1}{6}\sum_{(i,j,k)\in T_i \cap T_j} \mathrm{sgn}(i,j,k)(\mathbf{k}_i - \mathbf{k}_k)\cdot \hat{\mathbf{B}}.
\end{aligned}$$

Note that this matrix is antisymmetric and the elements on the main diagonal are equal to zero. The antisymmetry is apparent from the $\mathrm{sgn}(i,j,k)$ term. The diagonal being zero is a bit more subtle. Think about what the sum means when $i = j$. We are summing the edges not containing the point $i$ over all triangles containing $i$. But, for any point inside a closed surface, this just means going around the point $i$ in a closed loop. So, since we are concerned with closed surfaces, this is always zero.

Finally, for the in-scattering term, we have

$$\Xi_{ijmn} = \left(\int d\mathbf{k}\, \psi_i(\mathbf{k})\psi_m(\mathbf{k})\right)\left(\int d\mathbf{k}'\, \psi_j(\mathbf{k}')\psi_n(\mathbf{k}')\right) = M_{im}M_{jn} \tag{88}$$

$$S_{ij} = \sum_{m,n} \Xi_{ijmn} s^{mn} = \sum_{mn} M_{im}M_{jn}s^{mn}. \tag{89}$$

In summary, defining

$$\begin{cases} \alpha_i = \displaystyle\sum_{(i,j,k)\in T_i} \alpha(i,j,k), & (90) \\ \alpha_{ij} = \displaystyle\sum_{(i,j,k)\in T_i\cap T_j} \alpha(i,j,k), & (91) \\ \mathbf{d}_{ij} = \displaystyle\sum_{(i,j,k)\in T_i\cap T_j} \mathrm{sgn}(i,j,k)(\mathbf{k}_i - \mathbf{k}_k), & (92) \end{cases}$$

the different matrices can be expressed as

$$M_{ij} = \frac{\alpha_i}{12}\delta_{ij} + \frac{\alpha_{ij}}{24}\delta_{\langle ij\rangle}, \tag{93}$$

$$\Gamma_{ij} = \left(\frac{\alpha_i\gamma^i}{20} + \sum_k \frac{\alpha_{ik}\gamma^k}{60}\delta_{\langle ik\rangle}\right)\delta_{ij} + \left(\frac{\alpha_{ij}}{60}(\gamma^i+\gamma^j) + \sum_k \frac{\alpha(i,j,k)\gamma^k}{120}\delta_{\langle ik\rangle}\delta_{\langle jk\rangle}\right)\delta_{\langle ij\rangle} \tag{94}$$

$$(D_{\hat{\mathbf{v}}\times\hat{\mathbf{B}}})_{ij} = \frac{1}{6}\hat{\mathbf{B}}\cdot\mathbf{d}_{ij}\delta_{\langle ij\rangle}. \tag{95}$$

$$S_{ij} = \frac{\alpha_i\alpha_j}{12}s^{ij} + \alpha_i\sum_n \alpha_{jn}s^{in}\delta_{\langle jn\rangle} + \alpha_j\sum_m \alpha_{im}s^{mj}\delta_{\langle im\rangle} + \sum_{m,n}\alpha_{im}\alpha_{jn}s^{mn}\delta_{\langle im\rangle}\delta_{\langle jn\rangle}. \tag{96}$$

$\delta_{\langle ij\rangle}$ is equal to 1 if $i$ and $j$ are neighboring points, and 0 otherwise. Technically, due to the definitions, it is not really needed, but I include it to make things a bit clearer.

# 3 Implementation Details

## 3.1 Solving Matrix Equations

When sticking with the relaxation time approximation and ignoring the in-scattering, the matrices are sparse, since there is only overlap between neighboring elements. Even better, they are nearly *banded*, meaning that the nonzero elements are only on the main diagonal and the minor diagonals surrounding it. I say nearly, because there are also extra corner terms, coming from the periodic boundary conditions of the Brillouin zone.

For solving our matrix equations which involve nearly-banded matrices, there are two approaches. The first is to treat the matrix as a general sparse matrix and utilize sparse matrix solvers. The second approach is to use a banded matrix solver in combination with some other method for handling the off-diagonal, periodic boundary terms.

In terms of time complexity, efficient sparse matrix solvers achieve $\mathcal{O}(N^{3/2})$ performance, where $N$ indicates the number of nonzero elements in the matrix. In our systems, since the nonzero elements correspond to single-point terms and terms coming from the interaction of closest neighbors (which do not grow with the size of the system), $N$ scales linearly with the number of discretization points of the system. Note that this

is only true when there is no in-scattering, since that term generates fully dense matrices, because of the global interaction of states that the scattering Kernel introduces.

For a banded matrix solver, the time complexity is $\mathcal{O}(Nb^2)$ where $b$ is the *bandwidth* of the matrix, which is defined as the number of nonzero diagonals. So, whether this beats the sparse solver depends on the scaling of the bandwidth. For a 2D system, this bandwidth is constant and equal to 3, so our system is just a tridiagonal matrix, with 2 corner terms coming from the endpoints of the closed Fermi surface interacting with each other. This is extremely fast to solve, and thoroughly beats a sparse solver. However, for 3D systems, no matter how we index the points, the bandwidth of the matrix is about $\sqrt{N}$ and the time complexity will be around $\mathcal{O}(N^2)$, which is a worse scaling than the sparse solver.

To understand the 3D bandwidth scaling, picture a cylindrical surface. Imagine that to construct this surface, we stack rings of points on top of each other. To index the points, we number the points in each ring in order, and then move to the next layer. Each point, in addition to being neighbors with points that have close index numbers, is also neighbors with a point from the other end of the loop and also points from the top and bottom rings. The difference of the index of these other points from the point is around the number of points in a ring. Since there are $N$ points in total on the surface, the number of points in each of the rings should be around $\sqrt{N}$. The nonzero elements are belong to neighbors, and now we have neighbors that have indices that are $\sqrt{N}$ apart. On the matrix, this is a point where row number and the column number (which correspond to two different points) are $\sqrt{N}$ apart, which means that it is $\sqrt{N}$ away from the main diagonal. So, the bandwidth will be around $2\sqrt{N}$.

In practice, we see that the sparse solver is faster than the banded solver for any system size, not just large systems. See Figure 3. So, I decided to use a sparse solver in general.

## 3.2 Triangulation and Periodic Boundary Conditions

There are many different ways of dividing up the Fermi surface into triangles. I tried any of these triangulation methods that I could find, and I found that there is negligible difference between them. So, in the package, the simplest one is used, which is *marching cubes*. This was already used before in the Chambers package.

One difference of the FEM and the Chambers formula is the fact that in the FEM, the periodic boundary conditions of the Brillouin zone need to be enforced explicitly. This is only required if we have a Fermi surface that crosses the boundaries of the Brillouin zone, and the symmetry is broken in the axis where the crossing happens. So, for example, in the case of ADMR, where we have a rotating field, pointing in arbitrary directions.

To enforce the periodic boundary conditions, we first assume the triangulated Fermi surface has identical points on the opposite sides of the Brillouin zone on the axis where the crossing happens. Then, to force these points to be the same in the solution, we reduce the system to a smaller one with only unique points. Suppose we have some vector $\mathbf{b}$ where we have a quantity attributed to each point on the full triangulated Fermi surface. Now, the same quantity defined only on the unique points can be represented by a vector with smaller number of dimensions, $\mathbf{b}'$. To go from $\mathbf{b}$ to $\mathbf{b}'$, we can define a matrix $P$ such that

$$\mathbf{b}' = P\mathbf{b} \implies \mathbf{b} = P^T\mathbf{b}'. \tag{97}$$

Most of the columns in $P$ just contain a single 1, but for the points that are duplicated,

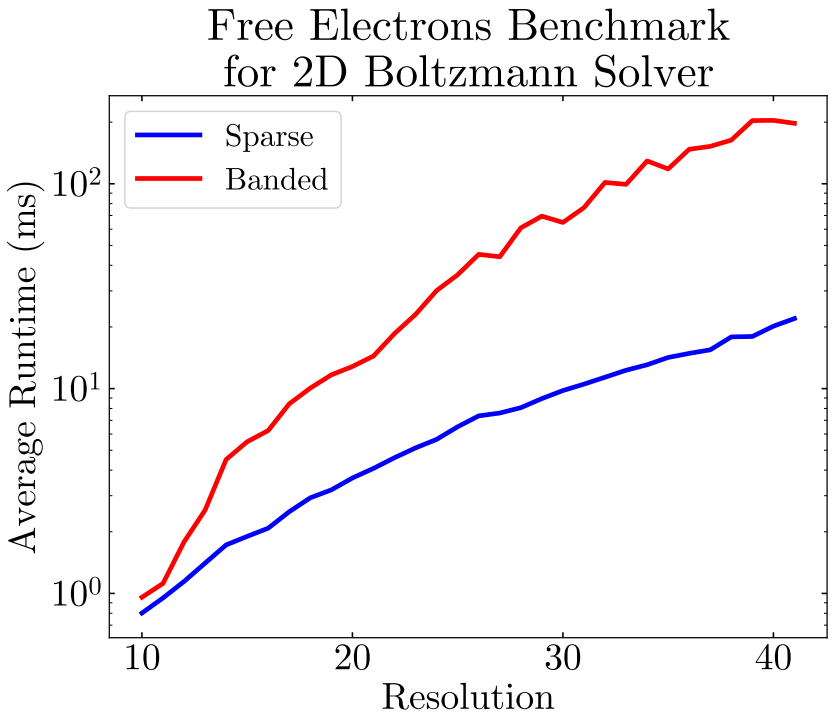


Figure 3: Comparison of the runtime of the sparse and banded implementations on a wide range of system sizes. The banded solver is never faster.

the corresponding column contains multiple 1s, one for each duplicate point. Now, if we have a matrix equation such that

$$A\mathbf{x} = \mathbf{b} \tag{98}$$

and we want the quantity

$$\sigma = \mathbf{a} \cdot \mathbf{x} = \mathbf{a} \cdot \left(A^{-1}\mathbf{b}\right), \tag{99}$$

we can rewrite everything in the "reduced" space as

$$A\mathbf{x} = P^T\mathbf{b}' \implies \sigma = P^T\mathbf{a}' \cdot \left(A^{-1}P^T\mathbf{b}'\right). \tag{100}$$

One might prefer to think of $P$ as a transformation for the matrix $A$:

$$\sigma = \mathbf{a}^T\mathbf{x} = \mathbf{a}'^T P A^{-1} P^T \mathbf{b}' = \mathbf{a}'^T (PAP^T)^{-1}\mathbf{b}', \tag{101}$$

but I like to keep it as a transformation for the vectors, since we can just transform the vectors and keep every other step the same. In fact. the transformation only needs to be done once, since $\mathbf{a}$ and $\mathbf{b}$ are different components of the projected velocity $\mathbf{v}_i$ and $\mathbf{v}_j$, and it suffices to transform the matrix containing each component in each column. This turns out to be faster than transforming the matrix $A$.

**Note:** In the code, $P^T$ is called the projection matrix and is all that the package works with, since it is the only thing that is needed to transform the vectors. You can think of $P^T$ as the transformation reducing the full space to the unique space. The reason I did not use this for the explanation is that it is not clear that the reduced space point should be a sum of the full space points before seeing the transformation matrix from the reduced space to the full space.

### 3.3 Curvature corrections

There is one very cheap correction we can do to improve the accuracy at low resolutions. When triangulating a curved surface, we are always underestimating the surface area, since the flat triangles always have less surface area than the patch on the curved surface that they are approximating. This happens when the points are on the surface. Now, imagine having the triangular faces tangent to the surface instead. In this case, the surface area is much closer to the actual curved surface. In the limit, these two approximations converge to the same surface, but at low resolutions, the tangent plane approximates the surface area much better. To understand this intuitively, imagine taking a curved patch of the surface and flattening it on a table. The resulting shape is the projection of the curved patch onto the tangent plane. This looks close to the tangent face we are using, but slight misalignments make it inexact. But it is still much better than the face that has the vertices on the surface.

## 4 Results

### 4.1 Free Electrons

Figure 4 shows the comparison of the FEM to the Drude model for free electrons in two dimensions. As expected, they match perfectly. The convergence is exponential, as seen in Figure 5. The results are quite similar in 3D, so I do not include them here.

### 4.2 Cuprates

In the more complicated case of the cuprates, we only have the Chambers formula to compare to. Since the Chambers formula itself is an approximation, it is hard to evaluate the actual accuracy of the FEM. However, seeing as the results of the FEM converge faster, and that the results of Chambers get closer to the results of the FEM at higher resolutions we can be confident that the FEM is more accurate. See Figure 6 and 7 for some comparisons. See Figure 8 for a comparison of the convergence. Also, you can see from the comparison to the Drude model in Figure 9 that the FEM is more accurate at calculating the resistivity for free electrons, which adds to the evidence that the FEM is more accurate.

We can also see that the $R_H$ of the FEM correctly converges to $-1/ne$ at very high fields. See Figure 10. This does not require high resolutions, and higher fields do not require more computation, unlike the Chambers formula. This shows another advantage of the FEM over the Chambers formula.

### 4.3 Performance

As seen in Figure 6 from the previous section, the FEM is 5 to 6 times faster than the Chambers formula at resolutions with similar accuracy. Keep in mind that this is a comparison to the marching squares implementation, which is faster but not always possible to use, as the absolute value of the resistivity can be quite inaccurate.

Having said that, the true power of the FEM becomes apparent in cleaner materials like $PdCoO_2$. The FEM can be two orders of magnitude faster. For an example, see Figure 11.

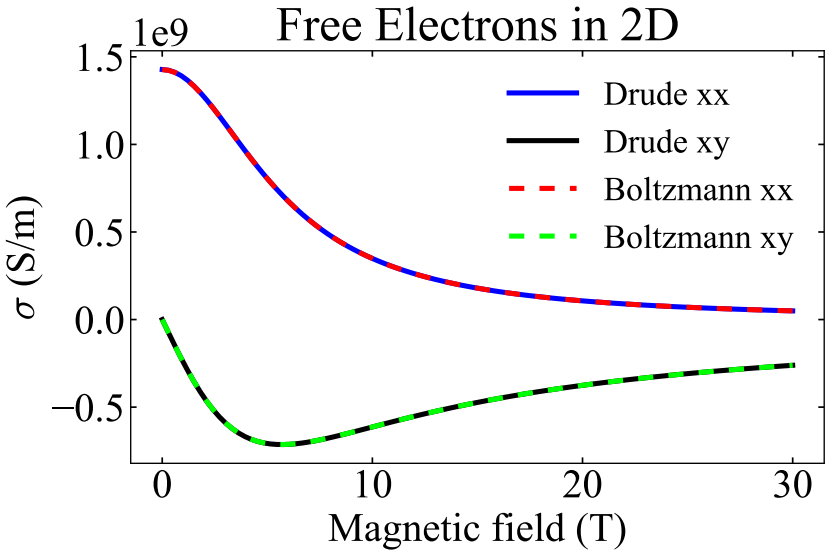


Figure 4: Comparison of the FEM to the Drude model for free electrons in 2D at a relatively low marching cubes resolution of 20.

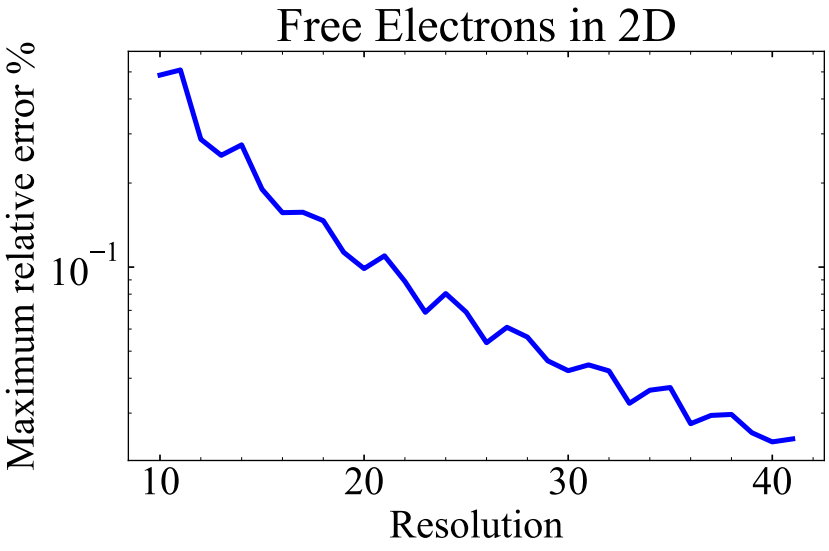


Figure 5: The maximum relative error of the FEM compared to the Drude model for free electrons in 2D as a function of the marching cubes resolution. The convergence is exponential, and we already hit 99.9% accuracy at a resolution of around 20. The jaggedness is because of imperfections in the triangulation.

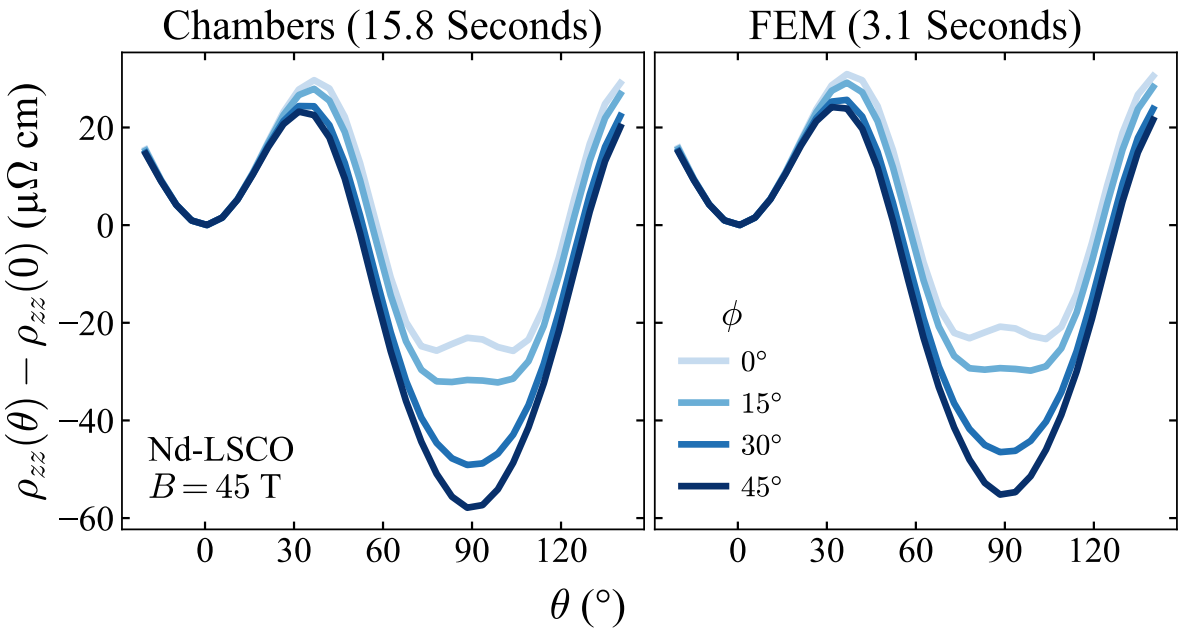


Figure 6: Comparison of the ADMR of Nd-LSCO calculated by the FEM and the Chambers formula. By focusing on the relative changes only instead of the full absolute resistivity, we can see that the models are quite similar.

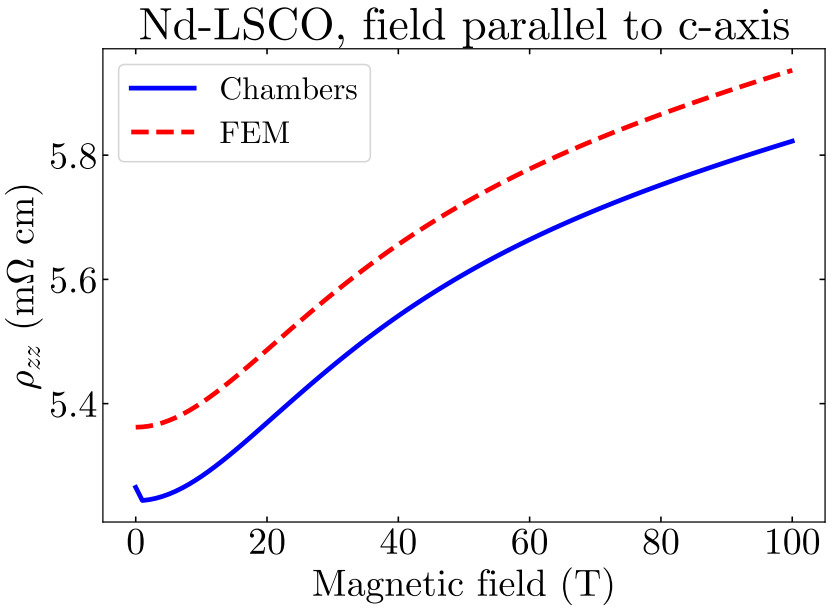


Figure 7: Comparison of the magnetoresistance of Nd-LSCO calculated by the FEM and the Chambers formula. This is at high resolution, and there is even a larger discrepency at lower resolutions. However, most of the discrepency comes from a difference in the absolute value, and the relative changes are quite similar.

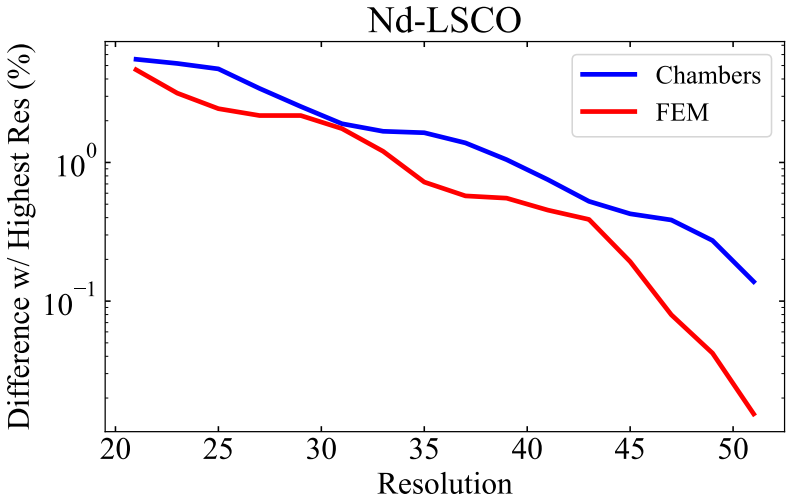


Figure 8: The convergence of the FEM and the Chambers formula for Nd-LSCO. As you can see, the FEM converges faster, which indicates that it is probably more accurate.

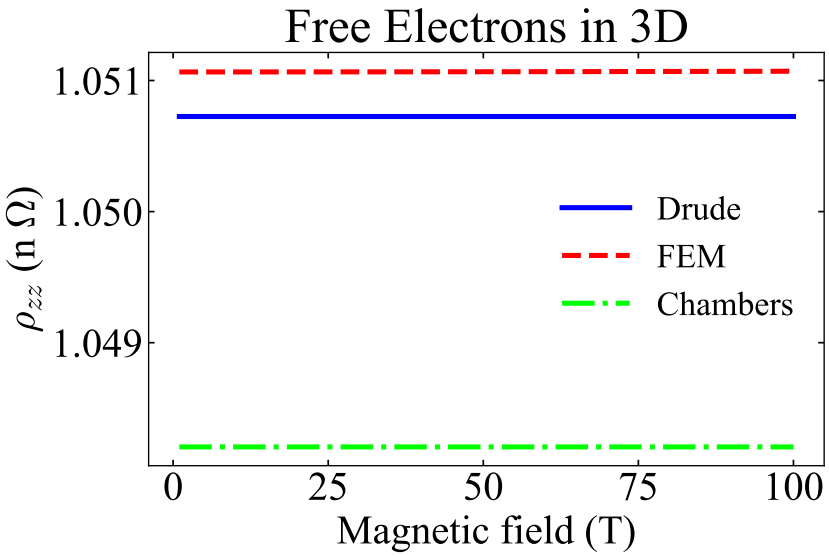


Figure 9: The resistivity of free electrons is 3D, as calculated by the FEM and the Chambers formula (with similar resolutions), compared to the Drude model that is the exact solution here. As you can see, the FEM is more accurate.

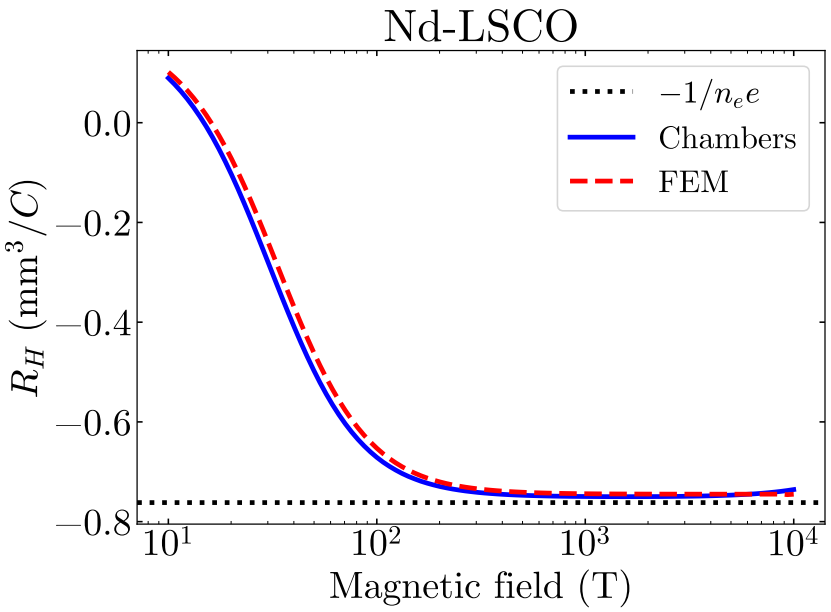


Figure 10: The Hall coefficient of Nd-LSCO calculated by the FEM and the Chambers formula. Both correctly converge to $-1/ne$ at high fields.

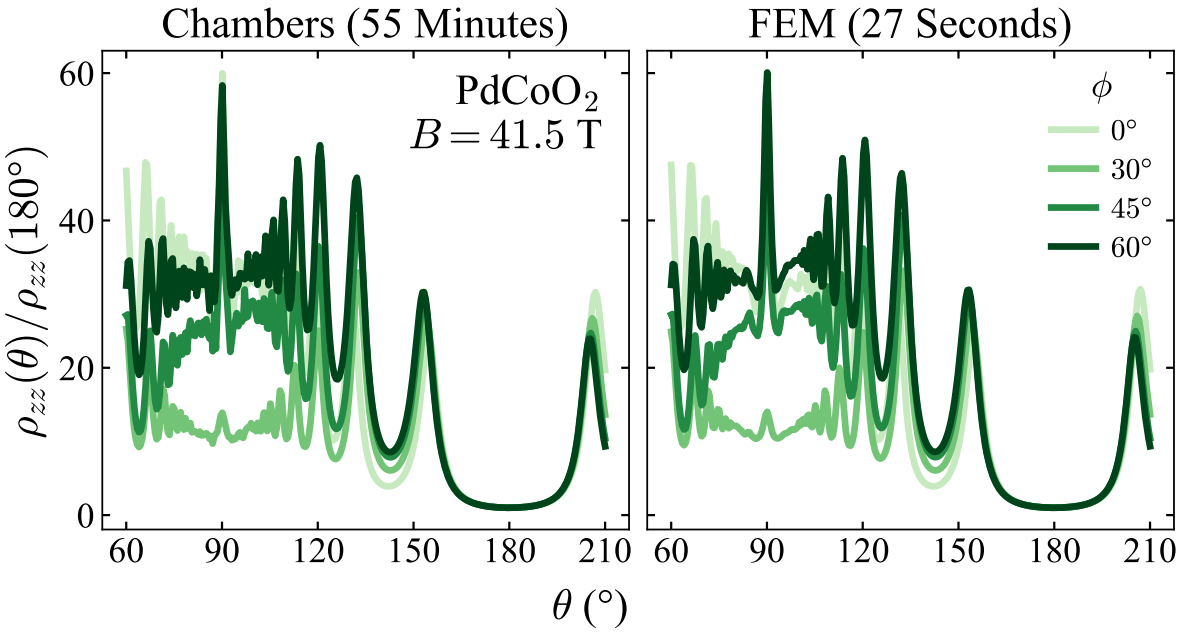


Figure 11: Comparison of the ADMR of $PdCoO_2$ calculated by the FEM and the Chambers formula. The FEM is more than 120 times faster at similar accuracy.