scdc

Release 0.0

Yonit Hochberg, Eric David Kramer, Noah Kurinsky, and Benjamin

Dec 21, 2020

CONTENTS

scdc stands for *quasiparticle Monte Carlo*. This code simulates the relaxation of quasiparticle excitations produced by an energy deposit in a superconductor, with particular attention to energy deposits from dark matter scattering. The code was developed to study directional correlations between the initial and final states, and is bundled with some simple tools to quantify that relationship.

The basic structure of the calculation is as follows: an energy deposit produces initial quasiparticles and/or phonons. The quasiparticles can relax by emitting phonons, and sufficiently energetic phonons can decay to quasiparticle pairs. This process continues until all quasiparticles are too low-energy to emit a phonon, and all phonons are too low-energy to produce a quasiparticle pair. These excitations constitute the final state, and we say that they are *ballistic*. The full set of excitations connecting initial to final states is generated as a tree of python objects, so every step of the shower can be inspected directly.

The main non-triviality is that the kinematical equations governing phonons and quasiparticles cannot be solved analytically, so the code solves them numerically.

CHAPTER

ONE

CONTENTS

1.1 Kinematics

We now review the kinematics of phonons and quasiparticles as treated by the code. Note that we work in the "clean" limit, where crystal momentum is conserved.

1.1.1 Units

In the code, we work in "material" units, setting $v_\ell = k_F = \Delta = 1$. This implies that $\hbar = 2\gamma\sqrt{z}$, where $z \equiv E_F/\Delta$.

1.1.2 Dispersion relations

The phonon dispersion relation is $E = \hbar q$. This is invertible: a phonon's energy uniquely determines its momentum. The same is not true for a quasiparticle, which has dispersion relation

$$E = \left[1 + \left(\frac{\hbar^2 k^2}{2m_e} - z\right)^2\right]^{1/2}.$$

The inverse is requires a choice of sign:

$$k = \frac{1}{\hbar}\left[2m_e\left(z \pm \sqrt{E^2 - 1}\right)\right]^{1/2}.$$

This corresponds to the fact that the energy is minimized ($E = \Delta$) at the Fermi surface, $k = k_F$, and increases in either direction. In the code, *if a quasiparticle is instantiated with only energy specified, we choose this sign randomly.*

1.1.3 Phonon decay to quasiparticles

A phonon can decay to quasiparticles if it has enough energy to produce a quasiparticle pair, i.e., $E > 2\Delta$. Such a decay is characterized by the energy ω of one of the two quasiparticles (and the azimuthal angle, which is always uniform-random). We take the differential rate in this variable from eq. (27) of Kaplan et al. (1976):

$$\frac{\mathrm{d}\Gamma}{\mathrm{d}\omega} = \frac{1}{\sqrt{\omega^2 - \Delta^2}} \frac{\omega\,(E-\omega) + \Delta^2}{\sqrt{(E-\omega)^2 - \Delta^2}}.$$

There is a complication here, however, because this gives the differential rate as a function of quasiparticle *energy*. As we just discussed, the energy does not uniquely determine the momentum of a quasiparticle, which is essential to ensure that we satisfy conservation of momentum. Since the decay rate into each allowed final-state momentum should be equivalent, we *randomly* select the sign in the inverse dispersion relation for each quasiparticle.

But this introduces yet another complication: it is possible that one or more of these sign combinations is not kinematically allowed, in that they can produce decays with $|\cos\theta| > 1$. Thus, we first select ω according to the above distribution, and then check whether each sign combination is kinematically allowed. We randomly select from the allowed combinations. These are called *candidate* final states.

1.1.4 Phonon emission by quasiparticles

1.2 Structure of the code

1.2.1 Events and overall structure of the calculation

The structure of the calculation is as follows.

A scattering event produces two quasiparticles. Each of those quasiparticles can subsequently relax by emitting phonons. If sufficiently energetic, those phonons can themselves decay to quasiparticle pairs. The process continues until none of the products have enough energy to undergo another scatter or decay.

This picture describes a tree of particles with interactions as the nodes of the tree, and we use a very similar structure to implement the calculation programmatically. Our tree consists of three types of object:

- Event
- Interaction
- Particle

In tree language, *Event* objects are nodes and *Particle* objects are edges. *Interaction* objects couple a single *Event* object to a collection of *Particle* objects.

Why not just have *Interaction* and *Particle* objects? Because a third type (*Event*) is required to specify the relationship between the two. The *Interaction* types certainly need to know about *Particle* types in order to create the final state of each interaction. But then how would a *Particle* know about the *Interaction* types for which it can be an initial state? Alternatively, how would each *Interaction* type know about all the others? At a technical level, this would require a circular import. At a conceptual level, it indicates a mutual dependency between *Particle* and *Interaction* types that deserves its own container.

So instead, the structure is as follows. The nodes in the tree are *Event* objects. *Particle* objects are the edges between *Event* objects. The *Event* objects "know" what *Interaction* types are possible for a given *Particle*, and they use such an *Interaction* object internally to produce a new final state. The *Event* can then create new *Event* nodes at which to terminate each final-state *Particle*.

Each *Interaction* can also be physically prohibited. When this happens, the "initial" state of such an *Interaction* is a leaf of the tree.

class scdc.event.**Event**(*initial_state*, *material*, **args*, ***kwargs*)

Interaction tree node.

Parameters

- **initial_state** (list of Particle) – particles in the initial state. The initial state is always listlike, even if it contains only one object.
- **(obj** (*material*) – *Material*): the material in which this takes place.

initial_state

particles in the initial state. The initial state is always a list, even if it contains only one object.

Type list of Particle

material (obj
Material): the material in which this takes place.

interaction
the interaction which can take place given the supplied initial state.

Type `Interaction`

out
particle *Event* objects.

Type `list` of `Particle`

final
True if no further interaction can take place.

Type bool

act()
Run the interaction and generate a final state.

chain()
Run the interaction and run particle interactions recursively.

property leaf_events
final events in the detector.

These are the leaves of the event tree, corresponding to the final states that will be measured.

Type `list` of *Event*

property leaf_particles
final particles after relaxation.

Type `list` of `Particle`

property out
List of out.

Type `list` of *Event*

1.2.2 Particle objects

class scdc.particle.**DarkMatter**(***kwargs*)
A dark matter particle.

dispersion(*p*)
Dispersion relation for dark matter.

Be careful: there is a need for conversion to the material's own canonical units. (NOT currently implemented.)

Parameters **p** (*float*) – momentum in who-knows-what units... to fix.

Returns dark matter energy in units of Delta.

Return type float

class scdc.particle.**Particle**(***kwargs*)
Represents a single phonon or quasiparticle with a tree of children.

momentum must be specified unless *dispersion_inverse* is defined, in which case either *momentum* or *energy* can be specified.

All quantities are specified in canonical units (v_l=k_F=Delta=1).

Parameters

- **energy** (*float, optional*) – energy in units of Delta.
- **momentum** (*float, optional*) – momentum in canonical units.
- **cos_theta** (*float*) – cosine of angle wrt global z.
- **material** (Material) – material properties.

energy
energy in units of Delta.

Type float

cos_theta
cosine of angle wrt global z.

Type float

material
material properties.

Type Material

parent
parent particle.

Type *Particle*

origin
the originating *Event* for this particle.

Type Event

dest
the destination *Event* for this particle.

Type Event

shortname
a short label for this particle type.

Type str

dispersion(*k*)
Disperion relation: wavenumber to energy.

dispersion_inverse(*E*)
Inverse dispersion relation: energy to wavenumber.

property energy
energy in units of Delta.

Type float

property momentum
momentum in canonical units (v_l=k_F=Delta=1).

Type float

class scdc.particle.**ParticleCollection**(**args*, ***kwargs*)
Represents a collection of particles.

Parameters **particles** (list of *Particle*) – particles.

particles
particles.

Type list of *Particle*

energy
energy of each particle.

Type ndarray of float

momentum
momentum of each particle.

Type ndarray of float

cos_theta
cos_theta of each particle.

Type ndarray of float

property cos_theta
array of particle *cos_theta* values.

Type ndarray

property dm
subcollection of DM particles.

Type *ParticleCollection*

property energy
array of particle energies.

Type ndarray

classmethod from_npy(*npy*, *material*)
Construct a *ParticleCollection* from the numpy representation.

Parameters

- **npy** (ndarray) – a structured array with columns as specified in the *to_npy* method.
- **material** (Material) – the material to associate with these particles. The material is not stored in the numpy format.

Returns a corresponding *ParticleCollection*.

Return type *ParticleCollection*

property nondark
subcollection of non-DM particles.

Type *ParticleCollection*

property phonons
subcollection of phonons.

Type *ParticleCollection*

property quasiparticles
subcollection of quasiparticles.

Type *ParticleCollection*

select(*test*)
Select a subset of particles which satisfy a test.

The argument *test* is a function with call signature

test(parent, particle)

which returns *True* if the leaf *particle* in the chain arising from the initial particle *parent* satisifes the test.

Parameters **test** (*function*) – the test function.

Returns

collection consisting of particles which satisfy the test.

Return type *ParticleCollection*

to_npy()

Represent this particle collection in numpy format for export.

The numpy format is a structured array with the following columns:

shortname momentum cos_theta

The numpy form REQUIRES momenta for all particles.

Returns the array form of this collection.

Return type ndarray

class scdc.particle.**Phonon**(***kwargs*)

A phonon.

dispersion(*k*)

Dispersion relation for phonons.

Parameters **k** (*float*) – wavenumber in canonical units (v_l=k_F=Delta=1).

Returns phonon energy in units of Delta.

Return type float

dispersion_inverse(*E*)

Inverse dispersion relation for phonons.

Parameters **E** (*float*) – phonon energy in units of Delta.

Returns phonon wavenumber in canonical units (v_l=k_F=Delta=1).

Return type float

class scdc.particle.**Quasiparticle**(***kwargs*)

A Bogoliubov quasiparticle.

dispersion(*k*)

Dispersion relation for quasiparticles.

Parameters **k** (*float*) – wavenumber in canonical units (v_l=k_F=Delta=1).

Returns quasiparticle energy in units of Delta.

Return type float

dispersion_inverse(*E*)

Inverse dispersion relation for quasiparticles.

Warning: the dispersion relation is not actually invertible. There are two possible choices of a sign. If the sign *s* is not specified, this method selects it randomly!

Parameters

- **E** (*float*) – quasiparticle energy.

- **s** (*int, optional*) – sign in the momentum solution.

Returns quasiparticle wavenumber in units v_l=k_F=Delta=1.

Return type float

property ksign
the sign as determined from energy and momentum.

Type int

1.2.3 Interaction objects

class scdc.interaction.**DarkMatterScatter**(**args*, ***kwargs*)
Represents an interaction in which DM scatters and creates two QPs.

This is a container, in the sense that no DM dynamics are actually contained here. Rather, at initialization, two quasiparticles must be supplied as arguments.

Parameters

- **qp1** (Quasiparticle) – final state quasiparticle 1.
- **qp2** (Quasiparticle) – final state quasiparticle 2.

allowed()
Determine whether the process is allowed for this initial state.

Returns *True* if the process can occur. Otherwise *False*.

Return type bool

class scdc.interaction.**Interaction**(*initial_state*, *material*, **args*, ***kwargs*)
Parent class for interactions.

An interaction combines an initial state, a material, and a final state. Subclasses must provide the means to go from the initial state to the final state.

Interactions are inherently treelike and can be treated as such. The complication is that the initial and final states are *Particle* objects, not *Interaction* objects. This necessitates the introduction of a new container type, *InteractionNode*, which specifies a particle and the interaction in which it participates.

Parameters

- **initial_state** (list of Particle) – particles in the initial state. The initial state is always a list, even if it contains only one object.
- **(obj** (*material*) – *Material*): the material in which this takes place.

initial_state
particles in the initial state. The initial state is always a list, even if it contains only one object.

Type list of Particle

material (obj
Material): the material in which this takes place.

n_initial
how many particles should be in the initial state.

Type int

n_final
how many particles should be in the final state.

Type int

initial
types of the initial particles, listed in order of their appearance.

Type tuple of *type*

final
types of the final particles, listed in order of their appearance.

Type tuple of *type*

ip
first of the initial particles.

Type Particle

allowed()
Determine whether the process is allowed for this initial state.

Returns *True* if the process can occur. Otherwise *False*.

Return type bool

interact()
Interact, validate outcome, and set final state.

classmethod valid_initial(*initial_state*)
Check that an initial state contains the correct particle types.

Parameters **initial_state** (list of Particle) – candidate initial state to validate.

Returns

***True* if the supplied initial state has the right particle** types for this interaction. Otherwise *False*.

Return type bool

class scdc.interaction.**PhononDecayToQuasiparticles**(**args*, ***kwargs*)
The *Interaction* for a phonon to decay to two quasiparticles.

Parameters **tolerance** (*float, optional*) – a numerical tolerance for root-finding. Defaults to 1e-3.

tolerance
a numerical tolerance for root-finding.

Type float, optional

allowed()
Determine whether the process is allowed.

Phonon decay can always take place if the energy of the phonon is at least 2*Delta.

Returns *True* if phonon decay can occur. Otherwise *False*.

Return type bool

decay_angles(*Eqp1*, *Eqp2*, *s1*, *s2*)
Computes the relative angles of two QPs created in phonon decay.

Parameters

- **Eqp1** (*float*) – energy of quasiparticle 1.
- **Eqp2** (*float*) – energy of quasiparticle 2.

- **s1** (*int*) – sign appearing in the inverse dispersion relation (QP1).
- **s2** (*int*) – sign appearing in the inverse dispersion relation (QP2).

Returns relative cos(theta) of quasiparticle 1. float: relative cos(theta) of quasiparticle 2.

Return type float

qp_energy_distribution(*n_bins=1000*, *eps=1e-10*)

Relative probability for a quasiparticle energy in phonon decay.

This is basically the integrand of eq. (27) of Kaplan+ 1976.

Parameters

- **n_bins** (*int, optional*) – number of (linear) bins to use for QP energies. Defaults to 1000.
- **eps** (*float, optional*) – slight offset to prevent divide-by-zero errors. Defaults to 1e-10.

Returns

QP energy bin centers. `ndarray` of `float`: relative rate for each bin center,

normalized to unity.

Return type `ndarray` of `float`

class scdc.interaction.**QuasiparticlePhononEmission**(**args*, ***kwargs*)

The *Interaction* for a quasiparticle to emit a phonon.

Parameters **tolerance** (*float, optional*) – a numerical tolerance for root-finding. Defaults to 1e-3.

tolerance

a numerical tolerance for root-finding.

Type float, optional

allowed(*exact=False*)

Determine whether the process is allowed for this initial state.

Returns *True* if the process can occur. Otherwise *False*.

Return type bool

final_state_angles(*phonon_energy*, *sp*)

Cosine of angle of final state particles after QP scattering.

Both angles are computed relative to the direction of the proximal quasiparticle just before scattering, not necessarily the initial quasiparticle.

DEBUGGING NOTE: this follows section "Better" of *dispersion.nb*.

Parameters

- **phonon_energy** (*float*) – phonon energy in units of Delta.
- **qp_energy** (*float*) – Initial quasiparticle energy in units of Delta.
- **sp** (`list` of int) – momentum sign for final quasiparticle.

Returns cosine of the angle of the emitted quasiparticle. float: cosine of the angle of the emitted phonon.

Return type float

min_cos_deflection()
Max angular deflection, expressed as minimum of cos(theta).

For a given QP energy, there is a maximum amount of momentum it can shed via phonon emission. This produces an upper bound on the angular deflection of the QP. We express this in the form of a lower bound on the cosine of the deflection angle.

Returns min of cos(theta).

Return type float

phonon_energy_distribution(*n_bins=1000*)
Relative phonon emission probability by a quasiparticle.

Computation follows dGamma/dx in eq. (21) of Noah's note. We normalize directly. If the distribution cannot be normalized (all values vanish), both return values are *None*.

Parameters **n_bins** (*int, optional*) – number of (linear) bins to use for phonon energies. Defaults to 1000.

Returns

phonon energy bin centers. ndarray of float: relative rate for each bin center,

normalized to unity.

Return type ndarray of float

Raises **NonPhysicalException** – If the max phonon energy is not positive.

phonon_energy_region()
Min / max allowed phonon energy for emission by a quasiparticle.

The min and max energies are determined by finding the energy range in which the final-state angles are physical, i.e. **|cos(theta)|** < 1. This needs to be done for both the quasiparticle and phonon angle. Moreover, since there is a sign ambiguity in the quasiparticle momentum, this needs to be done for each sign independently to get self-consistent regions that work for both the quasiparticle angle and the phonon angle.

Thus, we take the following approach. For each sign in turn, we find the physical region for each of the two angles. The intersection of these two regions is what we want. But then we have to do this for each sign, and take the union of the resulting intersections.

In principle, this may be disjoint. For the moment, rather than significantly change the architecture of the code, I will simply throw an error if the regions are disjoint.

Parameters **qp_energy** (*float*) – Initial quasiparticle energy in units of Delta.

Returns minimum emitted phonon energy in units of Delta. float: maximum emitted phonon energy in units of Delta.

Return type float

1.2.4 Material objects

class scdc.material.**Material**(**kwargs*)
Container class for material properties.

Parameters

- **symbol** (*str*) – chemical symbol.
- **gamma** (*float*) – gamma parameter.
- **c_s** (*float*) – speed of sound in m/s.
- **T_c** (*float*) – critical temperature in K.
- **Delta** (*float*) – half of the gap in eV.
- **E_F** (*float*) – Fermi energy in eV.
- **m_star** (*float*) – effective mass? in units of m_e.
- **beta** (*float*) – beta.

symbol
chemical symbol.

Type str

gamma
gamma parameter.

Type float

c_s
speed of sound in m/s.

Type float

T_c
critical temperature in K.

Type float

Delta
half of the gap.

Type float

E_F
Fermi energy in eV.

Type float

m_star
effective mass? in units of m_e.

Type float

z
Fermi energy in units of Delta.

Type float

m_e
electron mass in canonical units (v_l=k_F=Delta=1).

Type float

hbar
hbar in canonical units (v_l=k_F=Delta=1).

Type float

beta
beta.

Type float

mcs2
m*c_s^2 in units of eV.

Type float

qp_final_state_angles_approx(*phonon_energy*)
Cosine of angle of final state particles after QP scattering.

This version of the calculation makes the approximation that E_F is much, much larger than either the phonon energy or the quasiparticle energy. In this case, there is actually no dependence on the initial energy of the quasiparticle. I have checked that this approximation is very good in the regime we care about.

Parameters **phonon_energy** (*float*) – phonon energy in units of Delta.

Returns cosine of the angle of the emitted quasiparticle. float: cosine of the angle of the emitted phonon.

Return type float

1.3 Analysis tools

1.3.1 Analysis functions

This module defines functions for analysis of simulated ensembles.

scdc.analyze.**norm_asymmetry**(*particles*, *distance_function=<function p_dist.<locals>._dist>*, *n_bins=50*)
Find the asymmetry as the distance from an isotropic distribution.

For any norm, this will give zero for perfectly isotropic scattering. The maximum depends on the norm. For the default L^1 norm, if the scattering is purely directional, corresponding to a delta function, the difference will be 0.5 over the whole interval and then the delta function integrates to 1, so that'll be a maximum of 2. For the L^2 norm, the norm of the delta is not well defined.

** However, because of the default L^1 behavior, the result is divided by 2 regardless of the distance function. **

Because we're working with a histogram anyway, we don't want to use any actual quadrature. Instead, we want to use a p-norm of some kind. So the distance function here is not really a norm, but a single-point integrand thereof. For example, to use an L^2 norm, the distance function should be

lambda x: np.abs(x)**2

and the rest will be taken care of internally.

Parameters

- **particles** (ParticleCollection) – particles.
- **distance_function** (*function, optional*) – a function of one variable giving the integrand of the norm. Defaults to *p_dist(1)*.

- **n_bins** (*int, optional*) – number of bins to use for the norm. Defaults to 50.

Returns norm/2 of the distance from an isotropic distribution.

Return type float

scdc.analyze.**p_dist**(*p*)
Factory for L^p-norm integrands.

Parameters **p** (*float*) – p for the L^p norm.

Returns L^p norm integrand as a function of one argument.

Return type function

scdc.analyze.**plane_asymmetry**(*particles*, *n_bins=100*)
Find the asymmetry in a sliding bin of width 1 in cos(theta).

For cos(theta) = 1, this corresponds to forward-backward asymmetry.

Parameters **particles** (ParticleCollection) – particles.

1.3.2 Plotting functions

scdc.plot.**latex_exp_format**(*x*, *mfmt='%.3f'*)
Format a number in scientific notation in LaTeX.

Parameters

- **x** (*float*) – number to format.
- **mfmt** (*str, optional*) – format string for mantissa. Defaults to '%.3f'.

Returns LaTeX string (no $'s).

Return type str

scdc.plot.**tree_plot**(*particle*, *origin=0, 0*, ***kwargs*)
Plot all child scattering events.

Parameters

- **fig** (Figure, optional) – matplotlib figure object.
- **ax** (AxesSubplot, optional) – matplotlib axis object.
- **origin** (tuple of float, optional) – starting coordinates for the tree.
- **dm_color** (*str, optional*) – color for DM lines.
- **phonon_color** (*str, optional*) – color for phonon lines.
- **qp_color** (*str, optional*) – color for quasiparticle lines.
- **min_linewidth** (*float, optional*) – smallest linewidth (E = Delta).
- **max_linewidth** (*float, optional*) – largest linewidth, corresponding to the energy of the initial excitation.
- **max_linewidth_energy** (*float, optional*) – energy for max linewidth.
- **final_distance** (*float, optional*) – if specified, final (ballistic) state lines will be extended to this distance from (0, 0).
- **alpha** (*float, optional*) – opacity.

1.4 Using the code

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

q

INDEX

M

N

O

P

Q

S

T

V

Z