Documentation

Authors: Hanwen Jin, Simão Meneses João, Johannes Lischner

November 9, 2022

This document contains the documentation of our hot carrier generation code, as well as an explanation of the methods involved and physical interpretation of the objects being used.

Contents

Part I
Overview

1 Introduction

Metallic nanoparticles have a very special interaction with light. When light shines on a metal, conduction electrons oscillate back and forth driven by the electric field. This non-equilibrium configuration excites electrons into more energetic unoccupied states creating a very energetic electron-hole pair. These carriers can be very energetic (hot carriers) and can be harnessed for many applications. Plasmonic photocatalysis is one such example.

2 Fermi Golden Rule

Fermi's Golden Rule provides the rate of transitions to states with energy E due to a perturbation.

$$N_e\left(E,\omega\right)=\frac{2}{V}\sum_{if}\Gamma_{if}\left(\omega\right)\delta\left(E-E_f\right)$$

where

$$\Gamma_{if}=\frac{2\pi}{\hbar}\left|\left\langle f\left|\phi_{\text{tot}}\left(\omega\right)\right|i\right\rangle\right|^2\delta\left(E_f-E_i-\hbar\omega\right)f\left(E_i\right)\left(1-f\left(E_f\right)\right).$$

$N_e\left(E,\omega\right)dE$ is interpreted as the number of transitions per unit time with energy between E and $E+dE$ due to a perturbation of frequency ω. The interpretation of each of the terms is the following:

1. The electric field does not affect every pair of states in the same manner. The number $\left\langle f\left|\phi_{\text{tot}}\left(\omega\right)\right|i\right\rangle$ codifies this effect.

2. The product of Fermi functions $f\left(E_i\right)\left(1-f\left(E_f\right)\right)$ ensures that only initial states below the Fermi energy and final states above the Fermi energy contribute to this integral. A finite temperature relaxes this restriction slightly.

3. The second Dirac delta $\delta\left(E_f-E_i-\hbar\omega\right)$ is a consequence of the interaction with the electric field of frequency ω. The electric field only causes transitions which change the energy by $\hbar\omega$. In practice, there are many mechanisms going on inside the nanoparticle, which slightly violate this restriction, so the Dirac delta is instead approximated by a gaussian of a certain width γ, denoted by δ_γ

4. The first Dirac delta $\delta\left(E-E_f\right)$ means that only final states which have energy E are to be considered. Like before, this Dirac delta can also be approximated by a gaussian of a certain (different) width λ, representing the limited resolution of the measuring apparatus. If this is ignored, we have infinite resolution, which gives rise to sharp peaks.

Comment on units

In this document, these are the units being used

- Energy: eV
- Voltage: Volts
- Length: nanometers
- Time: second

So the physical constants have the following values:

- Planck's constant: $\hbar = 6.582 \times 10^{-16}$eV s $= 2.42 \times 10^{-17}$Ha s

For reference, the electric field E_{ref} used in experiments is such that the illumination intensity is $I = 1\text{mW}/\mu\text{m}^2$, so $E_{\text{ref}} = 8.7 \times 10^5\text{Volt/m} = 8.7 \times 10^{-4}\text{Volt/nm}$. For comparison, the average solar intensity on Earth's surface is 1360W/m^2, which is about 10^6 times weaker!

Electrostatic potential

The electric potential in the quasistatic approximation is found using COMSOL by defining a geometry and providing the dielectric constant at a certain frequency ω. COMSOL then provides the electrostatic potential at each point in space as a result. See the next section for more details.

Calculation of the Fermi Golden Rule

In practice, for large nanoparticles, the form used for the Fermi golden rule is the following:

$$N_e\left(E,\omega\right) = \frac{4\pi}{\hbar V}\left(1 - f\left(E\right)\right)\int_{-\infty}^{\infty} d\varepsilon f\left(\varepsilon\right)\delta_\gamma\left(E - \varepsilon - \hbar\omega\right)\Phi_\omega\left(\varepsilon, E\right) \tag{1}$$

where

$$\Phi_\omega\left(\varepsilon,\varepsilon'\right) = \text{Tr}\left[\phi_{\text{tot}}\left(\omega\right)\delta\left(\varepsilon - H\right)\phi^\dagger_{\text{tot}}\left(\omega\right)\delta\left(\varepsilon' - H\right)\right]$$

represents the optical matrix in energy space rather than the eigenbasis of the Hamiltonian. It is calculated by decomposing it into a double Chebyshev expansion

$$\Phi_\omega\left(\varepsilon,\varepsilon'\right) = \sum_{n=0}^{M-1}\sum_{m=0}^{M-1}\Delta_n\left(\varepsilon\right)\Delta_m\left(\varepsilon'\right)\mu_{nm} \tag{2}$$

where

$$\mu_{nm} = \text{Tr}\left[\phi_{\text{tot}}\left(\omega\right)T_n\left(H\right)\phi^\dagger_{\text{tot}}\left(\omega\right)T_m\left(H\right)\right] \tag{3}$$

is the matrix of Chebyshev moments. $T_n\left(H\right)$ is the n-th Chebyshev polynomial with a matrix argument.

3 Quick-start guide

The code is separated into four distinct sections, each for a distinct operation. The main sections are: 1) Generating the atomic positions inside the nanoparticle shape; 2) Computing the electric potential for those positions; 3) Calculating the Chebyshev moments with this potential and 4) Resumming the Chebyshev series to obtain the hot carrier generation rate. The following code can be found under the `examples` folder.

1. First, we need to generate the positions of the atoms within the nanoparticle. So we specify which shape we want and its size by calling for example

   ```
   julia src/generate_positions.jl sphere 2.0 positions.dat false
   ```

   The first argument is the shape. `sphere` can be replaced by `octahedron`, `cube` or `rhombic`" to produce other shapes. The meaning of the second argument depends on the shape. If the shape is a sphere, then the second argument is its radius in nanometers. If the shape is any of the other three options, it represents the length of any of their edges, in nanometers. The last argument is a flag which if set to true, will rescale the atomic positions so that the whole nanoparticle fits inside a cube of dimensions $3 \times 3 \times 3$nm. This is for technical purposes (see section on Comsol rescaling) and normally the flag should be set to false. The positions get written to a file specified by the third argument, here it was called `positions.dat`. This file is in a format which can be read by Comsol if needed. The first 9 lines of this file are the Comsol header. Following these lines, there are three columns, with the coordinates of the atoms inside the nanoparticle in units of nm. [images here]

2. Use the positions to generate the potential inside the nanoparticle. This requires the frequency of the incoming electric field as an input. If the nanoparticle is a sphere, the potential inside it has a known exact shape, which can be evaluated directly in Julia. If the shape isn't a sphere, then the potential has to be found by solving Maxwell's equations inside the nanoparticle. This is done with Comsol (see later sections for details on this). Assuming the nanoparticle is a sphere, we can run the following line to produce the potential

   ```
   julia src/potential_sphere.jl positions.dat potential.dat 1.9
   ```

   The first argument is the set of atomic positions inside the nanoparticle, the second argument is the name of the output file and the third argument is the frequency of the external electric field in units of eV, assuming the electric field is oriented along the z direction. The potential calculated is the following

   $$\phi_{\text{tot}}(\omega, z) = \frac{3\varepsilon_m}{\varepsilon(\omega) + 2\varepsilon_m}(-e)\, E_0 z$$

   where $E_0 = 1\text{V/nm}$, z is the position along the z axis in nanometers and $-e < 0$ is the electron charge. $\varepsilon_m = 2$ is the relative permittivity of the environment and $\varepsilon(\omega)$ is the (frequency-dependent) relative dielectric constant of the material used in the nanoparticle - gold in this case. The output file containing the potential consists of a 9-line header followed by four columns. The first three columns represent the x, y and z coordinates of the atoms (in nanometers) and the fourth column represents the complex electric potential at those points, in units of Volt.

3. Use the potential to generate the Chebyshev moment. Example:

   ```
   julia src/moments.jl potential.dat output.h5 sphere 2.0 1000 10
   ```

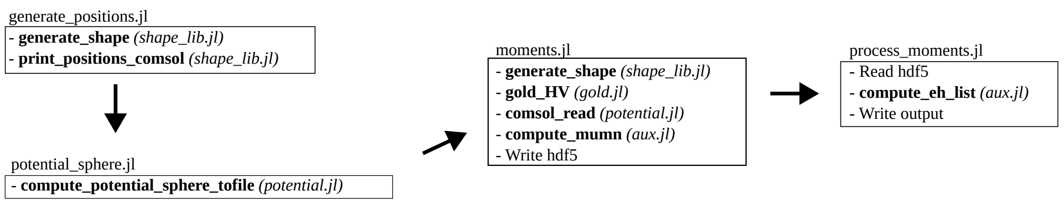
The first argument is the file with the electric potential, the second argument is the name of the file to which the Chebyshev moments will be written, the third argument is the shape, the fourth is the length associated with that shape (this has to be exactly the same value as used to obtain the atomic positions - see first point), the fifth is the number of Chebyshev polynomials to use in the expansion and the last argument is the number of random vectors used to compute the trace.

4. Use the Chebyshev moments to calculate the hot carrier generation rate. Example:

   ```
   julia src/process_moments.jl output.h5 rates.dat 1.9 0.2595 1000 70
   ```

   The first argument is the output from the previous point - the file with the Chebyshev moments. The second argument is the name of the file to be used as output with the hot carrier generation rates. The third argument is the frequency of the electric field (has to be identical to the one used in point 2) in units of eV. The fourth argument is the Fermi energy in units of Hartree. The fifth is the number of energies to be used both in the final numerical integration and as the number of energies at which the hot carrier generation rate is being calculated. The last argument is the percentage of Chebyshev moments to be kept (useful for convergence analysis). For example, if this is set to 70%, then 700 Chebyshev polynomials would be used instead of 1000 (see numbers of previous point). The output file has three columns: 1) Set of energies at which the hot carrier generation rate is being evaluated, in units of eV; 2) Hot electron generation rate (units of eV^{-1} s^{-1}); 3) Hot hole generation rate (units of eV^{-1} s^{-1}).

These four files are interfaces to the julia libraries containing the algorithms. The scheme below sums up the workflow and the functions (and libraries) envolved in this process. See next sections for more detail on each.


generate_positions.jl
- **generate_shape** *(shape_lib.jl)*
- **print_positions_comsol** *(shape_lib.jl)*
potential_sphere.jl
- **compute_potential_sphere_tofile** *(potential.jl)*
moments.jl
- **generate_shape** *(shape_lib.jl)*
- **gold_HV** *(gold.jl)*
- **comsol_read** *(potential.jl)*
- **compute_mumn** *(aux.jl)*
- Write hdf5
process_moments.jl
- Read hdf5
- **compute_eh_list** *(aux.jl)*
- Write output

4 Interpretation of the hot carrier generation rates.

To do

Part II

Technical information

5 Transforming the Fermi Golden Rule

Since diagonalization is very hard to do for large matrices, we need to express N_e in a different way in order to use a different set of methods. Let's begin by introducing two integrals over the energy in order to capture all the eigen-energies with Dirac deltas:

$$N_e\left(E,\omega\right)=\frac{2}{V}\sum_{if}\int_{-\infty}^{\infty}d\varepsilon\int_{-\infty}^{\infty}d\varepsilon'\delta\left(\varepsilon-E_i\right)\delta\left(\varepsilon'-E_f\right)\Gamma_{if}\left(\omega\right)\delta\left(E-E_f\right)$$

Now we just need to replace the eigen-energies:

$$N_e\left(E,\omega\right)=\frac{4\pi}{\hbar V}\int_{-\infty}^{\infty}d\varepsilon\int_{-\infty}^{\infty}d\varepsilon' f\left(\varepsilon\right)\left(1-f\left(\varepsilon'\right)\right)\delta\left(E-\varepsilon'\right)\delta\left(\varepsilon'-\varepsilon-\hbar\omega\right)\sum_{if}\left|\left\langle f\left|\Phi_{\text{tot}}\left(\omega\right)\right|i\right\rangle\right|^2\delta\left(\varepsilon-E_i\right)\delta\left(\varepsilon'-E_f\right)$$

which motivates us to define the new object

$$\Phi_\omega\left(\varepsilon,\varepsilon'\right)=\sum_{if}\left|\left\langle f\left|\Phi_{\text{tot}}\left(\omega\right)\right|i\right\rangle\right|^2\delta\left(\varepsilon-E_i\right)\delta\left(\varepsilon'-E_f\right).$$

This object is a reinterpretation of the potential matrix element in energy space. Instead of telling us the matrix element of the initial state i with final state f, it tells us the matrix element of initial state of energy ε with final state of energy ε'. It can be cast into a basis-independent form suitable for Chebyshev expansions:

$$\Phi_\omega\left(\varepsilon,\varepsilon'\right)=\text{Tr}\left[\Phi_{\text{tot}}\left(\omega\right)\delta\left(\varepsilon-H\right)\Phi_{\text{tot}}^{\dagger}\left(\omega\right)\delta\left(\varepsilon'-H\right)\right].$$

Plugging this back into the expression for the hot-carrier generation rate, we get

$$N_e\left(E,\omega\right)=\frac{4\pi}{\hbar V}\int_{-\infty}^{\infty}d\varepsilon\int_{-\infty}^{\infty}d\varepsilon' f\left(\varepsilon\right)\left(1-f\left(\varepsilon'\right)\right)\delta\left(E-\varepsilon'\right)\delta\left(\varepsilon'-\varepsilon-\hbar\omega\right)\Phi_\omega\left(\varepsilon,\varepsilon'\right).$$

$\Phi_\omega\left(\varepsilon,\varepsilon'\right)$ is simply a function of two energies, so $N_e\left(E,\omega\right)$ can be obtained through a straightfoward numerical double integral. This form also makes it easier to interpret what is happening to $\Phi_\omega\left(\varepsilon,\varepsilon'\right)$ during the integration process. Using the first Dirac delta to eliminate the integral in ε', the hot carrier generation rate can be simplified to

$$N_e\left(E,\omega\right)=\frac{4\pi}{\hbar V}\left(1-f\left(E\right)\right)\int_{-\infty}^{\infty}d\varepsilon f\left(\varepsilon\right)\delta_\gamma\left(E-\varepsilon-\hbar\omega\right)\Phi_\omega\left(\varepsilon,E\right). \tag{4}$$

The final ingredient to understand the inner workings of the code is the Chebyshev expansion the Dirac delta operators inside of $\Phi_\omega\left(\varepsilon,\varepsilon'\right)$. These operators can be expanded in a series of Chebyshev polynomials

$$\delta\left(\varepsilon-H\right)=\sum_{n=0}^{M-1}\Delta_n\left(\varepsilon\right)T_n\left(H\right)$$

where

$$\Delta_n\left(\varepsilon\right)=\frac{2}{\delta_{n,0}+1}\frac{T_n\left(\varepsilon\right)}{\pi\sqrt{1-\varepsilon^2}}g_J^{n,M}$$

is the coefficient of the expansion regularized by the Jackson weight g_J. Thus, $\Phi_\omega(\varepsilon,\varepsilon')$ is expressed as

$$\Phi_\omega(\varepsilon,\varepsilon') = \mathrm{Tr}\left[\Phi_{\mathrm{tot}}(\omega)\left(\sum_{n=0}^{M-1}\Delta_n(\varepsilon)T_n(H)\right)\Phi^\dagger_{\mathrm{tot}}(\omega)\left(\sum_{m=0}^{M-1}\Delta_m(\varepsilon')T_m(H)\right)\right]$$

and can be calculated in two steps. The first step is to calculate the Chebyshev moments

$$\mu_{nm} = \mathrm{Tr}\left[\Phi_{\mathrm{tot}}(\omega)T_n(H)\Phi^\dagger_{\mathrm{tot}}(\omega)T_m(H)\right]$$

and the second is to resum the matrix:

$$\Phi_\omega(\varepsilon,\varepsilon') = \sum_{n=0}^{M-1}\sum_{m=0}^{M-1}\Delta_n(\varepsilon)\Delta_m(\varepsilon')\mu_{nm}$$

6 Calculating the electric potential: Comsol

To find the electric potential inside the nanoparticle, we use Comsol to generate a simulation box representing the boundary conditions. We assume that the cube has side length L and is centered at the origin. In order to generate a uniform electric field inside the nanoparticle, we impose boundary conditions $V(z) = E_0 z$ on the faces of the cube. If z is in units of nm and $E_0 = 1$ Volt/nm, then V is in units of Volt. Finally, to convert from an electric potential to an electric potential energy $\Phi_0 = |e|V_0 = 1\mathrm{eV}$.

With these boundary conditions, we obtain an electric potential $V(\mathbf{r})$ which depends nontrivially on the position because of the dielectric properties of the nanoparticle. Because of the boundary conditions imposed, away from the nanoparticle, we expect $V(|\mathbf{r}|\to\infty) = E_0 z$. If the simulation box is sufficiently large, we expect the result not to depend on its size.

6.1 Rescaling the electric field

By design, the electric field inside the simulation box is always $E_0 = 1$ V/nm. To analyze other electric fields, we make use of the linarity of Maxwell's equations: if the electric field is twice as strong, then the induced electrostatic potential (in the quasistatic approximation) is twice as strong as well. Therefore, we can state that for a generic electric field strength E, the corresponding potential V_E is

$$V_E(\mathbf{r}) = \frac{E}{E_0}V_{E_0}(\mathbf{r})$$

so we can always use the default value of E_0 so long as we rescale it back to E in the end. With this in mind, the electric potential energy term inside the expression for N_e (eq. 1) can always be calculated with the default electric field in Comsol. Let Φ^0 be the energy-resolved optical matrix with this potential. Then

$$N_e(\varepsilon,\omega) = \frac{4\pi}{\hbar V}\left(\frac{E}{E_0}\right)^2(1-f(\varepsilon))\int_{-\infty}^{\infty}d\varepsilon f(\varepsilon)\delta_\gamma(E-\varepsilon-\hbar\omega)\Phi^0_\omega(\varepsilon,\varepsilon).$$

6.2 Rescaling the nanoparticle

Suppose that we want to find the electric potential inside several identically-shaped nanoparticles, but with different sizes. For example, cubic nanoparticles of different side lengths. We can of course use larger simulation boxes for larger particles in order to mitigate boundary effects, but there's a trick that can be used: rescale the nanoparticles into having identical sizes and rescale the value of the electric potential by the same amount.

For example, suppose we want to find the electric potential inside a nanocube of length $\ell = 3$nm and another of length 20nm. Let's further assume that we have already found the solution $V_{\ell=3}(\mathbf{r})$ for all positions inside the smaller nanocube. Then, $V_{\ell=3}(\mathbf{r}) = \frac{20}{3} V_{\ell=20}\left(\frac{3}{20}\mathbf{r}\right)$. More generally, any two potential profiles are related through

$$aV_{\ell=a}\left(\frac{\mathbf{r}}{a}\right) = bV_{\ell=b}\left(\frac{\mathbf{r}}{b}\right).$$

This is what is used in our implementation.

6.3 Using Comsol

Details on using Comsol

7 Chebyshev method

Part III
Code

8 External dependencies

These are the dependencies required to run the program.

- julia with the following dependencies
  - DelimitedFiles
  - Printf
  - MKL
  - Random
  - Dates
  - MKLSparse
  - HDF5
  - NearestNeighbors
  - DataStructures
  - Interpolations
  - SparseArrays
  - LinearAlgebra

Lines to install them using the Julia prompt

```
using Pkg;
Pkg.add(["DelimitedFiles", "Printf", "MKL", "Random", "Dates", "MKLSparse", "HDF5", "N
```

- python3 with vpython for visualization

9 Code organization [To do]

This program consists of a main code written in Julia (.jl) and several auxiliary programs in Python.

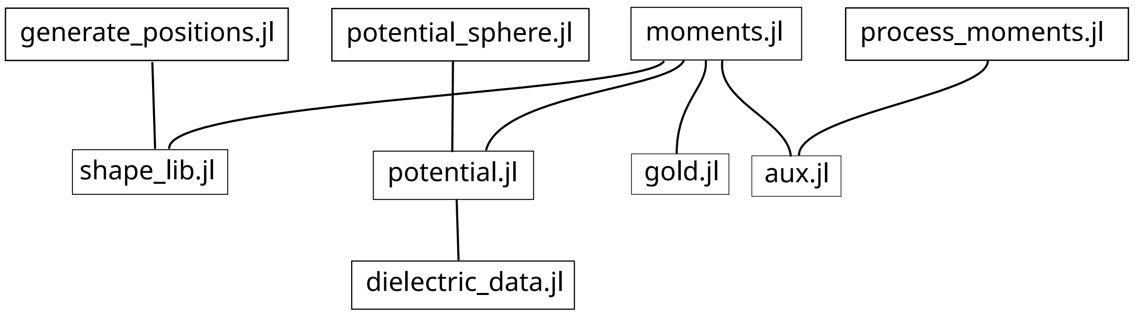


Figure 1: Diagram showing how the code is organized.