Calculating the Harmonic Response of Gas Particles Using the SFA Approximation

King's College London

6CCP3131 Third Year Project

Alex Benitez

Supervisor: **Amelle Zaïr**

Abstract: Machine Learning for image recognition requires large datasets to be trained. High Harmonic Generation is a very computationally expensive process to simulate, inhibiting the ability to train Neural Networks without experimental data. This report outlines an Open-Source package called SNAIL, which greatly speeds up calculation of the HHG spectrum generated by various types of particles and customizable strong fields.

Contents

Theoretical Background

1.1 The Three Step Model

Most nonlinear optical phenomena follow a simple relationship where the magnitude of the response drops linearly or exponentially with the order of the effect. In 1993 a series of experiments [5][7] delivered a very surprising result, using high intensity lasers, a gas medium would generate a response with harmonics (pulses with multiples of the driving laser frequency) up to two orders of magnitude of the driving pulse, far exceeding any intuitive understanding. The simplest way to explain this result is with the three-step model, the origin of which is debated but generally attributed to [1][2][3]. In the model, a strong laser pulse deforms the potential well of an electron in an atom, the electron tunnels into the continuum at 0 velocity under the influence of the electric field, during a laser cycle it travels away and comes back to the atom, where it can scatter or recombine. Focusing only on the recombination, the electron will emit a pulse, equivalent to the kinetic energy gained in the continuum. This simple model allows for an intuitive understanding of most of the effects observed in High Harmonic Generation (HHG).

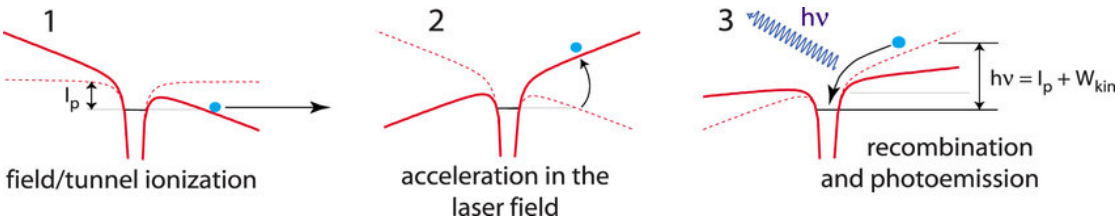

Figure 1.1: Graphic picturing the three step model of HHG, reprinted from [10]

Experimentally the maximum harmonic was found to be $I_p + 3U_p$ where I_p is the ionization potential of the atom and $U_p = \frac{e^2E^2}{4mw_o^2}$ is the ponderomotive energy, which is just the time averaged kinetic energy. Using the three step model, the maximum kinetic energy of the electron can be modelled, by considering a series of electrons under the influence of a simple laser the maximum kinetic energy is found to be around 3.17 U_p which will have I_p added to it as soon as the electron recombines, closely matching experimental results.

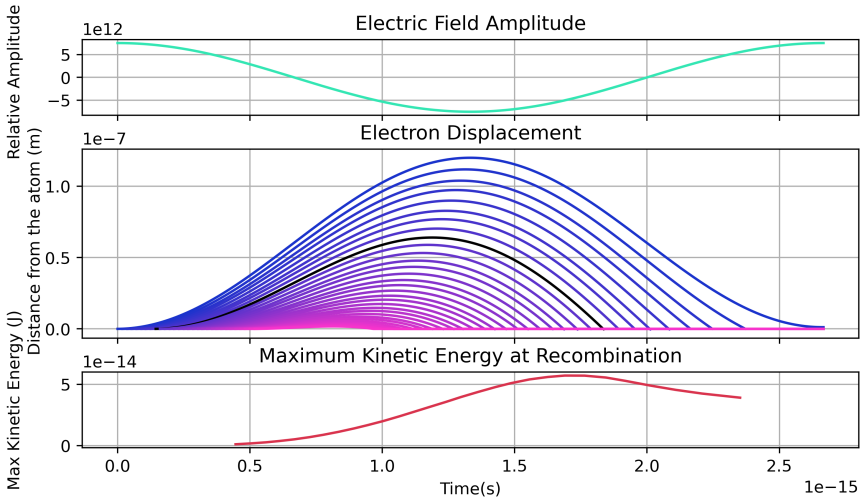

Figure 1.2: Plot demonstrating the 1D trajectories of electrons released during positive electric field values. From top to bottom, the driving field, distance of the electron from the nucleus and the maximum kinetic energy at recombination. In the second plot the trajectory with the highest kinetic energy is black.

1.2 The Strong Field Approximation

In 1994, Maciej Lewenstein and others introduced one of the most powerful tools to calculate HHG in a landmark paper [6]. The main assumptions of the Lewenstein model are: a) The ground state does not deplete, b) The ground state is the only bound state that affects the evolution of the system, c) The free electron only feels the external field, ignoring the atomic potential. For the ground state to not deplete, the ponderomotive force should be below U_{sat}, the saturation potential where all of the atoms ionize during interaction. Following the method set out in [9], [4] and [8], for the remainder of the text except where explicitly stated we use atomic units. We start with the TDSE of one shell electron:

$$i\frac{\partial}{\partial t}|\psi(t)\rangle = \left[-\frac{1}{2}\nabla^2 + V(r) - Ecos(t)x\right]|\psi(t)\rangle = \hat{H}(t)|\psi(t)\rangle \tag{1.1}$$

Here V(r) is the atomic potential, $Ecos(t)$ is the time-varying electric field from the driving pulse and $\hat{H}(t)$ is the Hamiltonian. What we want to find is the induced dipole, which is proportional to the response and given by:

$$\mathbf{D}(t) = \langle\psi(t)|\hat{\mathbf{d}}|\psi(t)\rangle \tag{1.2}$$

To calculate this, however, we need to represent the wavefunction in a convenient form; so we introduce propagators. A possible solution for (1.1), is:

$$|\psi(t)\rangle = e^{-i\int_0^t \hat{H}(t')dt'}|\psi(0)\rangle = e^{-i\int_0^t \hat{H}(t')dt'}|g\rangle \tag{1.3}$$

Where we set the initial state of the wavefunction as $|g\rangle$ implying the electron starts in the ground state in the atomic well. However, this form is not actually useful; the Hamiltonian can be split into any two parts, which we choose as $\hat{H} = \hat{H}_0 + \hat{V}_L$. In this case $\hat{H}_0$ is the Hamiltonian of the atomic potential, and $\hat{V}_L$ is the interaction with the laser. To split it, we have to find an expression for the propagator of our full Hamiltonian, which we will call $U_1(t,t')$, first we assume:

$$i\frac{\partial}{\partial t}U_0(t,t') = H_0U_0(t,t') \qquad U_0(t,t) = 1 \tag{1.4}$$

and we want to solve:

$$i\frac{\partial}{\partial t}U(t,t') = (H_0 + \Delta H)U(t,t') \qquad U(t,t) = 1 \tag{1.5}$$

The Dyson expansion states that this can be found recursively, and only taking the first term of the expansion since we neglect interaction with the atomic potential:

$$U(t,t') = U_0(t,t') \; - \; i\int_{t'}^{t} dt''U(t,t'')\Delta H(t'')U_0(t'',t') \tag{1.6}$$

A quick way to prove this is to substitute it in for itself which assumes knowledge of U_0 and U being a solution of itself and uses the Leibniz integral rule in the second line:

$$\begin{aligned} i\frac{\partial U(t,t')}{\partial t} &= i\frac{\partial}{\partial t}\left[-i\int_{t'}^{t} dt''U(t,t'')\Delta H(t'')U_0(t'',t') + U_0)t,t')\right] \\ &= U(t,t)\Delta H(t)U_0(t,t') + \int_{t'}^{t} dt''i\frac{\partial U(t,t'')}{\partial t}\Delta H(t'')U_0(t'',t') + i\frac{\partial}{\partial t}U_0(t,t') \\ &= (H_0+\Delta H)\int_{t'}^{t} dt''U(t,t'')\Delta H(t'')U_0(t'',t') + \Delta HU_0(t,t') + H_0U_0(t,t') \\ &= (H_0+\Delta H)U(t,t') \end{aligned} \tag{1.7}$$

Finally, substituting in back for the exponential terms, the wavefunction can be expressed as:

$$|\psi(t)\rangle = e^{-i\int_0^t \hat{H}_0(t'')dt''}|g\rangle \; - \; i\int_0^t dt' \left[e^{-i\int_{t'}^t \hat{H}(t'')dt''}\right]\hat{V}_L(t')\left[e^{-i\int_0^{t'} \hat{H}_0(t'')dt''}\right]|g\rangle \tag{1.8}$$

This very scary equation is surprisingly intuitive, the first standalone exponential term is the evolution of the non-ionized part of the electronic wavefunction. Inside the integral, the first exponential term acting on the ground state represents the evolution of the electron before it ionizes $\hat{H}_0(t'') = \hat{\mathbf{p}}^2/2 + U(\hat{\mathbf{r}})$; the $\hat{V}_L$ term indicates the transition from the ground state to the continuum, which happens at t'. The last term indicates the evolution and interaction with the electric field, which is why it includes the full Hamiltonian $\hat{H}$.

Equation (1.8), is still not very useful, since we still cant solve for the full Hamiltonian, here is where the assumptions of SFA shine. First, we assume that the Hamiltonian when the electron is in the laser field is given by $\hat{H}_F = \hat{H} - \hat{V}_A$, meaning we ignore the atomic potential. Luckily, the propagator for this case is known as the Volkov propagator, which when applied to an electron appearing in the continuum at t' with momentum $\mathbf{p}$ and defining $\mathbf{A}(t) = -\int \mathbf{F}(t)\,dt$:

$$\begin{aligned} \hat{U}_V(t,t')|\mathbf{p}+\mathbf{A}(t')\rangle &= e^{-iS_V(\mathbf{p},t,t')}|\mathbf{p}+\mathbf{A}(t)\rangle \\ S_V(\mathbf{p},t,t') &= \frac{1}{2}\int_{t'}^t dt''[\mathbf{p}+\mathbf{A}(t'')]^2 \\ \langle \mathbf{r}\,|\,\mathbf{p}+\mathbf{A}(t)\rangle &= \frac{1}{(2\pi)^{3/2}}e^{i[\mathbf{p}+\mathbf{A}(t)]\cdot\mathbf{r}} \end{aligned} \tag{1.9}$$

The physical explanation of this equation is, the electron enters the field from t' to t with momentum $\mathbf{p}$, by t, it has a momentum of $\mathbf{p}+\mathbf{A}(t)$ and a phase term of $e^{-iS_V(\mathbf{p},t,t')}$. From (1.9), we know that the coordinate of the wavefunction is a plane wave, which at each moment in time forms a complete basis, meaning:

$$\int d\mathbf{p}|\mathbf{p}+\mathbf{A}(t)\rangle\langle\mathbf{p}+\mathbf{A}(t)| = \hat{1} \tag{1.10}$$

We can finally calculate the dipole over time, making the final assumptions that there is no permanent dipole in the ground state and the electron doesn't scatter or absorb extra energy from the field non-adiabatically:

$$\mathbf{D}(t) \approx -i\langle\hat{U}_0(t,t_0)g\,|\,\hat{\mathbf{d}}|\int_{t_0}^t dt'\hat{U}_V(t,t')\hat{V}_L(t')\hat{U}_0(t',t_0)|g\rangle + c.c. \tag{1.11}$$

Using identity (1.10), and expanding some terms, this takes the form of:

$$\begin{gathered} \mathbf{D}(t) = i\int_{t_0}^t dt''\int d\mathbf{p}\,\mathbf{d}^*(\mathbf{p}+\mathbf{A}(t))e^{-iS(\mathbf{p},t,t')}\mathbf{F}(t')\mathbf{d}(\mathbf{p}+\mathbf{A}(t')) + c.c. \\ where \quad \mathbf{d}(\mathbf{p}+\mathbf{A}(t)) = \langle\mathbf{d}(\mathbf{p}+\mathbf{A}(t))\,|\,\hat{\mathbf{d}}|\,g\rangle \end{gathered} \tag{1.12}$$

Here the total action is used which includes the Ionization potential I_p. One last rearrangement gives us the form:

$$\begin{gathered} \mathbf{D}(t) = i\int_{t_0}^t dt''\int d\mathbf{p}\,\mathbf{d}^*(\mathbf{p}+\mathbf{A}(t))e^{-iS(\mathbf{p},t,t')}\Phi(\mathbf{p}+\mathbf{A}(t'')) + c.c. \\ \Phi(\mathbf{p}+\mathbf{A}(t'')) = \left[\frac{(\mathbf{p}+\mathbf{A}(t''))^2}{2} + I_p\right]\langle\mathbf{p}+\mathbf{A}(t'')\,|\,g\rangle \end{gathered} \tag{1.13}$$

The final step to get the harmonic spectrum is to calculate the Fourier transform:

$$I(N\omega) \propto (N\omega)^4|D(N\omega)|^2 \tag{1.14}$$

1.3 The Saddle Point Method

The previous section provided us with a very useful approximation, but if we were to try to calculate this, we would soon run into large rounding errors since the action term makes the integral highly oscillatory. The action oscillates much quicker than the rest of the terms in the integral, this means that the biggest contributions to the integral (1.12) are when the action varies slowly, since the response of the rest of the trajectories will destructively interfere. Additionally, due to our assumptions, the action is the difference in position of the electron between two different times, which leads to stationary points:

$$\nabla S(\mathbf{p},t,t') = \mathbf{x}(t) - \mathbf{x}(t') = 0 \tag{1.15}$$

This tells us that the most 'relevant' trajectories are the ones that begina and end in the same place, which are the electrons that recombine. The action integral can be expanded:

$$\begin{aligned} S(\mathbf{p},t,t') &= \int_{t_0}^{t} dt'' \left(\frac{(\mathbf{p}-\mathbf{A}(t''))^2}{2} + I_p \right) \\ &= \frac{1}{2}\mathbf{p}^2(t-t') - \mathbf{p}\cdot\int_{t_0}^{t} \mathbf{A}(t'')dt'' + \int_{t_0}^{t} dt'' \left(\frac{\mathbf{A}(t'')^2}{2} + I_p \right) \end{aligned} \tag{1.16}$$

Here, the second term is 0, the integral of a potential over a closed loop is always 0, and our electron ends where it starts. The third term can be taken out of the momentum integral since it only depends on the field. Finally we can also assume the momentum integration will mostly be affected by the action integral and the dipole elements can be taken out:

$$i\int_{t_0}^{t} dt'' \, \mathbf{d}^*(\mathbf{p}+\mathbf{A}(t))\mathbf{d}(\mathbf{p}+\mathbf{A}(t'))\mathbf{F}(t')e^{-i\int_{t_0}^{t} dt'' \left(\frac{\mathbf{A}(t'')^2}{2} + I_p \right)} \int d\mathbf{p} e^{-\frac{1}{2}i\mathbf{p}^2(t-t')} + c.c. \tag{1.17}$$

The last integral is very similar to a gaussian integral, and if we convert from integrating over return time, to integrating over total excursion time $\tau = t - t'$, and add a small factorε to the exponent making it $exp(-i\frac{1}{2}p^2(\tau - i\varepsilon))$ it becomes a full gaussian, without losing much accuracy if the ε is small. The saddle point approximation states that we can take integrals of this form as:

$$\int F(\beta)e^{\rho\phi(\beta)}d\beta = \sum_s \sqrt{\frac{2\pi}{\rho}} \frac{F(\beta)e^{\rho\phi(\beta)}}{\sqrt{-\phi''(\beta_s)}} \tag{1.18}$$

Where the summation is over the saddle points, taking our saddle points that start and end in hte same place, we can finally write our integral in the form:

$$\mathbf{D}(t) = -\int_0^{\infty} d\tau \left(\frac{\pi}{\varepsilon + i\tau/2} \right)^{3/2} \mathbf{F}(t')\,\mathbf{d}(\mathbf{p}(t,\tau)+\mathbf{A}(t'))\;\mathbf{d}^*(\mathbf{p}(t,\tau)+\mathbf{A}(t))e^{-iS(t,\tau)} + c.c. \tag{1.19}$$

Where the action no longer depends on momentum, and the integral oscillates much more slowly. This integral can once again be explained intuitively, the prefactor accounts for the spread of the returning wavepacket, which can classically be thought of as the electron having lower chance of recombining the longer the excursion time. The field and the first dipole give the probability that the electron ionizes, and the second dipole is the probability of recombination. Finally, the last term is the action phase acquired by the electron during the excursion.

SNAIL: An SFA library

The aim of this project was to develop an open-source python library which can numerically integrate (1.19). I aimed to make it as computationally efficient as possible, to allow for Machine Learning (ML) applications and landscape studies, where several factors in a simulation are slowly changed to reveal novel effects. It is fully implemented on Python, and based on HHGMax**??**.

2.1 Integration Parameters

One final problem we encounter is (1.19) has an infinite integration limit, however, we know that the prefactor will make any long excursions have a negligible effect, therefore we can set our integration limit to a large value without losing much accuracy. There is still a problem, where an arbitrary limit will create artifacts due to the oscillatory nature of the integral, to remedy this a 'soft' cos^2 window is applied past a certain τ_d, until $\tau_{max} = 0$; a good τ_{max} value is 1.5 driving field cycles, which will capture the major contributions, however the window can be increased for further accuracy. We also have to define the ground state wavefunction, in [6] a scaled hydrogen-like potential is used, which takes into account the I_p of the target.

We are now equipped to begin integrating, after all our hard work, we can just use the trapezium rule. However, simply plugging in (1.19) into a regular Python script will be incredibly slow, this is because Python 'for' loops carry out extra operations of no use to us, which increases the complexity to O(n^2), without taking into account the actual operations.

2.2 Vectorization With Numpy

Numpy is a versatile Python library whose main benefit to us is a highly efficient implementation of matrices and matrix operations, which we will call arrays from now on. The main difference is Numpy allows for operations between arrays without the use of for loops; arrays are also stored in contiguous blocks of memory as opposed to Python lists which are stored as pointers, greatly increasing the access time. The drawback is finding ways to implement all of our calculations as matrix operations. To illustrate how to do this and the speedup, I will use the example of calculating $\mathbf{A}(\mathbf{t})$ from the driving field $\mathbf{E}(t)$, as stated before $\mathbf{A}(t) = -\int \mathbf{E}(t)\,dt$. Traditionally, calculating this in Python would look like this:

```
1 import math
2 t = [i*2*3.14159/1000 for i in range(1000)]
3 Et = [math.cos(i) for i in t]
4 At = [0 for i in range(1000)]
5 for i in range(1,1000):
6         dt = t[i] - t[i-1]
7         temp = (Et[i] + Et[i-1])*dt/2
8         At[i] = temp
```

The key steps in the for loop are: i) finding the time interval dt (for non evenly spaced t), ii) adding the current and previous values of the electric field multiplied by dt, iii) Assigning this value to the predefined At. Within each iteration of this loop, in addition to the already slow inbuilt for loop, there are 6 accesses to memory and 3 definitions.

The key to vectorization is to find operations which require access from adjacent elements of a list at each step. For example, all of the values dt will take can be expressed as subtracting the array t with all the values shifted forward by 1, from the array t. A nice visualization of this is:

t [i]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
											-										
t [i-1]	20	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
											=										
dt [i]	-20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Here we have to be very careful with the elements at the beginning and the end of the array, in this case Numpy shifts the last element of the array to the front, meaning the first element of A(t) is unphysical. Luckily for us, whenever this happens, we can almost always set those elements to 0, since they are either actually 0 like in this case, or represent the electron before it ionizes. Using this principle and the function np.roll(), which shifts the values of an array by the specified number:

```
t = np.linspace(0,2*np.pi,1000)
Et = np.cos(t)
dt = (-np.roll(t,1) + t)*0.5
At = -(np.roll(Et,1) + Et)*dt
At[0] = 0
At = np.cumsum(At)
```

If we run both of these scripts 10000 times, not including imports since they are a negligible fraction of SNAIL's runtime, Numpy takes 0.77 seconds, whilst regular Python takes 10.5. Such a simple case is already an order of magnitude faster, and a more complicated implementation offers more chances for Numpy to speed up the code. One last trick for a speedup is generating large lists where each row is shifted forward by a different index, this can be done as follows:

```
ws = weights.size
bigAt = np.reshape(np.tile(At,ws-1),(ws-1,At.size))
temptAt = bigAt[np.c_[:bigAt.shape[0]] ,
(np.r_[:bigAt.shape[1]] - np.c_[1:ws]) % bigAt.shape[1]]
```

This starts with np.tile, which repeats an array n times, here, it creates an array with At repeated ws-1 times, and then reshapes it into the form (ws-1,N), where N is the number of timesteps. This shape of (ws-1,N) is the basic shape of our array, where rows are trajectories with increasing excursion times and columns are timesteps forward. The reason why it is only ws-1 is because the first line is excursion times of 0 which are ignored. The next line generates an array where each line is shifted forward by one more than the previous, the first expression np.c_[:bigAt.shape[0]] creates a column vector from 0 to ws. In the second expression, np.r_[:ws] generates a row vector which has a column vector subtracted from it; this generates a (N,ws) matrix which has the modulus taken by its own height, quickly creating the matrix we need, it can be visualized as:

		np.c_[1:ws]		
np.r_[bigAt.shape[1]]		[0]		[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
		[1]		[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8],
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]	-	[2]	=>	[-2, -1, 0, 1, 2, 3, 4, 5, 6, 7],
		[3]		[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6],
		[4]		[-4, -3, -2, -1, 0, 1, 2, 3, 4, 5]]

2.3 The Lewenstein Integral

With this technique in hand, we can begin to calculate the integral, the code begins by defining a set of useful arrays, namely At, $B(t) = \int dt A(t)$ and $C(t) = \int dt A(t)^2$:

```
At = -(np.roll(Et,1) + Et)*dt
At = np.squeeze(At) # This removes extra dimensions that might arise from improper formatting
At[0] = 0
At = np.cumsum(At)
Bt = (np.roll(At,1) + At)*dt
Bt[0] = 0
Bt = np.cumsum(Bt)
Ct = (np.square(np.roll(At,1)) + np.square(At))*dt
Ct[0] = (0,0,0)
Ct = np.cumsum(Ct)
```

Afterwards we calculate the momentum and both dipoles:

```
ws = weights.size
c = (np.pi/(epsilon_t + 0.5*1j*t[:ws]))**1.5
pst = (bigBt - temptBt)/np.c_[t[1:ws]]
correction = np.r_[:pst.shape[1]] > np.c_[:pst.shape[0]]
pst = pst*correction
argdstar = pst - bigAt
argdstar = argdstar*correction
argdnorm = pst - temptAt
argdnorm = argdnorm*(correction)

dstar = np.conjugate(dp(argdstar))
dnorm = dp(argdnorm)
dnorm = dnorm*correction
dstar = dstar*correction
```

First are the weights, an array storing the multiplicative factor of the integration window mentioned earlier, this is automatically calculated according to the parameters set by the user. Afterwards, we calculate the prefactor c, which will be stored in an array as big as the weights, meaning each integration step has a different c. Then we calculate pst, which is the momentum for a given excursion time. Correction is a matrix to eliminate parts of the trajectories which are unphysical as previously mentioned. It is created similarly to the row-shift matrices, except it is filled with boolean values rather than integers, which avoids issues with complex-valued matrices. After this we calculate the action, and finally the integral:

```
SQR = np.square
Sst = -(0.5/np.array([t[1:ws]]).T)*SQR(bigBt - temptBt)
        + 0.5*(bigCt-temptCt) + Ip*np.reshape(t[1:ws],(1,ws-1)).T
del temptBt
del temptCt
Sst = Sst*correction
integral = dstar*dnorm*np.exp(-1j*Sst)*temptEt*(np.c_[weights[1:]])
                *(np.c_[c[1:]])*(bigat)*temptat
output = np.cumsum(integral,0)[-1]
```

2.4 Auxiliary functions and parallelization

The Lewenstein module is the critical part of the library, but the bulk of the code is found in the auxiliary functions, these give the user the necessary tools to create the beam and set the parameters for the experiment. This part of the project required discussion with researchers on the best way to format the code for the wider community. The main functions are:

config This is the class that stores all of the user defined variables throughout the code. The decision to use a class rather than pass variables when calling functions comes from many variables, like the wavelength, being used in different parts of the code, and different use cases requiring some variables and not others.

sau_convert All of the math is carried out in scaled atomic units (SAU), which vary depending on the driving pulse and target, this function provides the tools to convert SI units to those used by the code to increase ease of access and mitigate unit errors.

generate_pulse This function gives the user common driving field types and many options to tweak them, however, many users will want to generate their own fields; using a custom field only requires converting it to SAU.

dipole_response If lewenstein() is the engine, this is the rest of the car. This function takes the driving field and user set variables, sets up the integration interval, and passes all the necessary values to lewenstein(). It multiplies the response by $cos(t/2)^2$, and returns the fourier transform (FT). The multiplication reduces noise in the FT, and the FT allows direct observation of the harmonic spectrum.

This report focuses on the lewenstein module, which runs on a single core, however, most users applying SNAIL to ML or landscape studies will have access to High Performance Computing (HPC). Due to this, there is a parallel_lewenstein module which, as the name implies, allows the code to run in parallel over several cores. The code does this by finding the variables common to different processes, and making them available to each thread, then calculating a different response on each core; this speeds up regular calculation linearly with the number of cores, but memory increases slower due to the array caching.

Bibliography

[1] F. Brunel. "Harmonic Generation Due to Plasma Effects in a Gas Undergoing Multiphoton Ionization in the High-Intensity Limit". In: *JOSA B* 7.4 (Apr. 1, 1990), pp. 521–526. ISSN: 1520-8540. DOI: 10.1364/JOSAB.7.000521. URL: https://opg.optica.org/josab/abstract.cfm?uri=josab-7-4-521 (visited on 03/14/2025).

[2] F. Brunel. "Not-so-Resonant, Resonant Absorption". In: *Physical Review Letters* 59.1 (July 6, 1987), pp. 52–55. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.59.52. URL: https://link.aps.org/doi/10.1103/PhysRevLett.59.52 (visited on 03/14/2025).

[3] P. B. Corkum, N. H. Burnett, and F. Brunel. "Above-Threshold Ionization in the Long-Wavelength Limit". In: *Physical Review Letters* 62.11 (Mar. 13, 1989), pp. 1259–1262. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.62.1259. URL: https://link.aps.org/doi/10.1103/PhysRevLett.62.1259 (visited on 03/14/2025).

[4] Misha Yu Ivanov, Michael Spanner, and Olga Smirnova. "Anatomy of Strong Field Ionization". In: *Journal of Modern Optics* 52.2–3 (Jan. 20, 2005), pp. 165–184. ISSN: 0950-0340, 1362-3044. DOI: 10.1080/09500340420002753 60. URL: http://www.tandfonline.com/doi/abs/10.1080/09500340420002753 60 (visited on 03/20/2025).

[5] Anne L'Huillier and Ph. Balcou. "High-Order Harmonic Generation in Rare Gases with a 1-Ps 1053-Nm Laser". In: *Physical Review Letters* 70.6 (Feb. 8, 1993), pp. 774–777. DOI: 10.1103/PhysRevLett.70.774. URL: https://link.aps.org/doi/10.1103/PhysRevLett.70.774 (visited on 03/18/2025).

[6] M. Lewenstein et al. "Theory of High-Harmonic Generation by Low-Frequency Laser Fields". In: *Physical Review A* 49.3 (Mar. 1, 1994), pp. 2117–2132. DOI: 10.1103/PhysRevA.49.2117. URL: https://link.aps.org/doi/10.1103/PhysRevA.49.2117 (visited on 03/13/2025).

[7] J. J. Macklin, J. D. Kmetec, and C. L. Gordon. "High-Order Harmonic Generation Using Intense Femtosecond Pulses". In: *Physical Review Letters* 70.6 (Feb. 8, 1993), pp. 766–769. DOI: 10.1103/PhysRevLett.70.766. URL: https://link.aps.org/doi/10.1103/PhysRevLett.70.766 (visited on 03/18/2025).

[8] Emilio Pisanty. *Under-the-Barrier Electron-Ion Interaction during Tunnel Ionization*. Aug. 7, 2013. DOI: 10.48550/arXiv.1307.7329. arXiv: 1307.7329. URL: http://arxiv.org/abs/1307.7329 (visited on 03/21/2025). Pre-published.

[9] Olga Smirnova and Misha Ivanov. *Multielectron High Harmonic Generation: Simple Man on a Complex Plane*. Apr. 8, 2013. DOI: 10.48550/arXiv.1304.2413. arXiv: 1304.2413 [physics]. URL: http://arxiv.org/abs/1304.2413 (visited on 03/13/2025). Pre-published.

[10] Carsten Winterfeldt, Christian Spielmann, and Gustav Gerber. "Colloquium: Optimal Control of High-Harmonic Generation". In: *Reviews of Modern Physics* 80.1 (Jan. 2, 2008), pp. 117–140. DOI: 10.1103/RevModPhys.80.117. URL: https://link.aps.org/doi/10.1103/RevModPhys.80.117 (visited on 03/15/2025).