FeyPy: A Python package for visualizing and calculating nonlinear optical properties

Adam Goler, Washington State University
adam.goler@email.wsu.edu

July 28, 2014

1 Features

FeyPy is an object-oriented Python package. The programming objective for this package was to construct a framework for visualizing complicated nonlinear optical processes using a Feynman diagram approach and then to use those visualizations to calculate useful properties of such systems, such as the overall nonlinear response of a series of molecules interactive via optical cascading. Other useful things, such as LaTeX strings and the ability to fit data to an expected nonlinear process are also implemented.

2 Package Directory

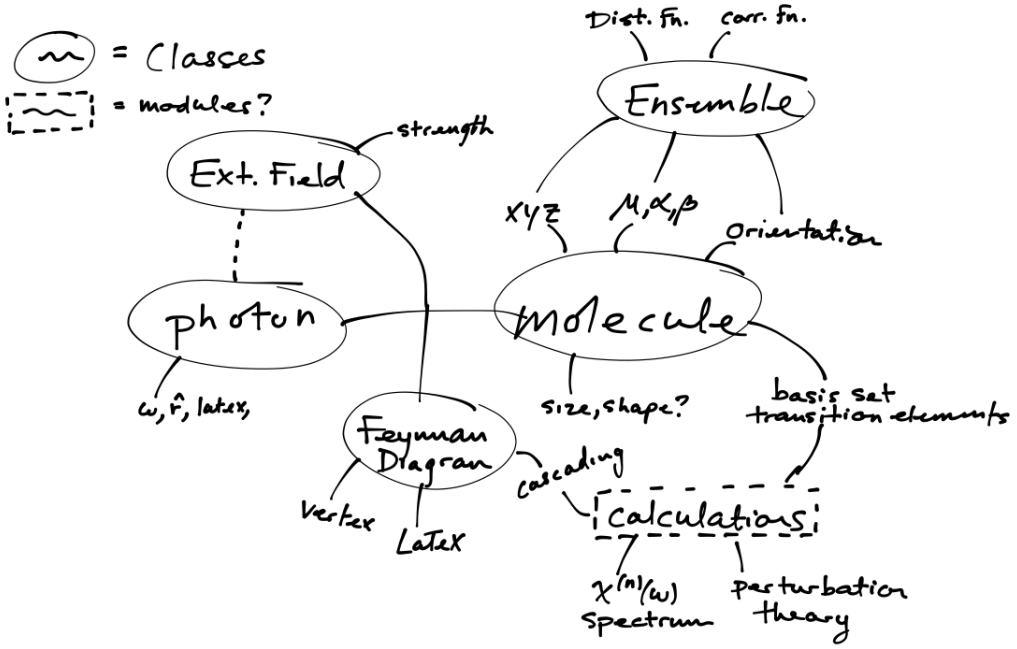

Figure 1: Hypothetical bubble chart indicating possible program structure and implied use relationships.

3 Getting Started

As this packages is programmed in Python, Python is required for running any of the following calculations. FeyPy is compatible with Python v2.7 or later. In addition, Numpy 1.8 (or later) and Matplotlib is also required.

To install FeyPy, simply download and/or extract all source code to a single directory and run with a Python interpreter. At this point, FeyPy is not configured to install directly to your Python path. Expect this to change in later iterations of the code.

The current version of FeyPy is v.0.2.

4 Examples

The utility of FeyPy is evidenced by its versatility. It is intended to be a one-stop-shop for visualization, calculation and fitting of nonlinear processes to the mind's eye and to experimental data. The following examples should serve as an effective introduction to the basic workings of the code. For questions about specific functions, or for a description of each available function (some are cleverly hidden to avoid accidental malarkey), see the Function Glossary (work in progress).

4.1 Constructing a simple Feynman diagram

Before beginning, be sure that all required packages (outlined in Getting Started) are installed. To import, make the following calls in your interpreter:

```
from Feynman import *
from Photon import *
from Permutation import *
from Molecule import *
from CalcChi import *
from Cascade import *
# eventually, hope to have this simply be replaced with
# from FeyPy import *
import numpy as np
```

Note any errors you receive when attempting to import the above; they could indicate improper installation of required packages. A Feynman diagram is represented by a Feynman object within FeyPy, or, alternatively, an object of class Feynman. Such an object is defined as follows:

```
feyn = Feynman()
```

In general, there are two types of particles depicted by Feynman diagrams: photons, and molecules that are absorbing or emitting photons. Defining a single Feynman diagram (object) represents the interaction of a single molecule with an arbitrary number of photons. Within FeyPy, photons are treated as objects of the class Photon. In order to define a photon, one needs to input the frequency and polarization as basic information. (Note: Other arguments can be used when defining photons, which will be described later.)

In this example, three photons will be defined.

```
photon1 = Photon(1,0)        # the first argument is frequency,
photon2 = Photon(1,0)        # the second argument is polarization
photon3 = Photon(-2,0)
```

To calculate meaningful results, make sure the output frequency (thus, energy) is the negative sum of the input frequencies. The polarization should be defined as either 0, 1, or 2, representing photons polarized along x, y, or z, respectively. Units are defined such that $\hbar = 1$

Once the photons are defined, all that remains is to add the photons to our Feynman diagram by using the *addPhoton()* command, which takes individual photon objects as arguments. Implement as follows:

```
# for a small number of photons, it might be easier to add individually
feyn.addPhoton(photon1)
feyn.addPhoton(photon2)
feyn.addPhoton(photon3)
```

```
# --> OR <-- for a set of photons, it may be easier to add using a for loop
photon_list = [photon1, photon2, ... photonN]
[feyn.addPhoton(photon_list[i]) for i in range(len(photon_list))]
```

This is the basic information needed to construct a Feynman diagram visualization. See the following sections for visualization methods and other calculations.

4.2 Viewing a Feynman diagram

Given the Feynman diagram object *feyn* defined in the previous section, one may visualize the diagram with the following code:

```
feyn.showDiagram()
```

As defined, this call results in the Feynman diagram shown in Fig. 2 (YMMV).

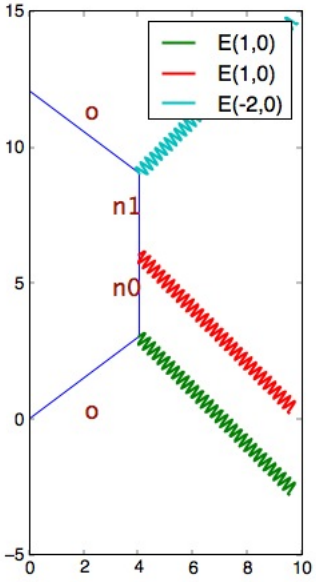

Figure 2: Example Feynman diagram illustrating a χ^2 nonlinear process.

4.3 Obtaining LaTeX code for a Feynman diagram

In addition to simple visualizations of Feynman diagrams, LaTeX code corresponding to the equation for the total nonlinear response can be retrieved either as plain text or shown using Matplotlib. In order to return the plain text code, simply use

```
feyn.getLaTeX()        # return TeX code as plain text
# returns: '\\chi^{2}=\\sum_{all}\\frac{\\mu^{0}_{gn_1}\\mu^{0}_{n_1n_0}
# \\mu^{0}_{n_0g}}{(\\Omega_{n_0g}-\\omega_{1})(\\Omega_{n_1g}-2\\omega_{1})}'
```

which, when copy/pasted into your favorite LaTeX editor (and with the removal of extra errant backslashes), results in Eq. 1

$$\chi^{(2)} = \sum_{all} \frac{\mu^0_{gn_1}\mu^0_{n_1n_0}\mu^0_{n_0g}}{(\Omega_{n_0g} - \omega_1)(\Omega_{n_1g} - 2\omega_1)} \tag{1}$$

It is also possible to view Eq. 1 without the use of a LaTeX compiler within the context of the *showDiagram()* command as follows:

```
feyn.showDiagram(latex=True)
```

4.4 Finding permutations of a Feynman diagram

The single diagram defined previously is not the whole story. Eq. 1 refers to a sum over potentially many Feynman diagrams, which represent unique permutations of the input/output photons so defined. FeyPy can not only determine what these unique permutations are, but it can also generate Feynman objects for each permutation which can themselves be interacted with using any method defined for Feynman objects.

To generate these permuted Feynman diagrams, use the following code:

```
list_of_feynmans = feyn.getPermutedFeynmans()
```

Each element of the defined list is itself a Feynman object, so each can be asked to generate an independent Feynman diagram, as shown in Fig. 3.

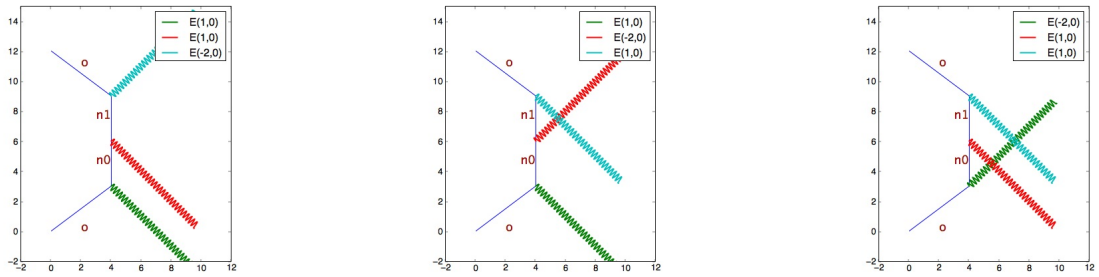

Figure 3: All unique permutations of the example Feynman diagram.

4.5 Calculating the nonlinear response for a single diagram and its permutations

Beyond the utility of visualizing Feynman diagrams, FeyPy is also capable of calculating nonlinear susceptibilities of individual diagrams or even the total nonlinearity of a set of all possible Feynman diagrams. For the Feynman object already defined as a result of the previous examples, we must first define a set of transition elements (Mu's) and resonant frequencies (Omega's) in order to calculate the nonlinear response. We use the *Molecule* class to achieve this. Objects of class Molecule represent molecules who's interaction with photons are represented by Feynman diagrams (and hence Feynman objects). Besides transition elements, and transition energies, molecule objects also contain position and orientation information relative to the lab frame. This information is necessary for cascading calculations, which will be addressed later. When dealing with single molecules, position and orientation is irrelevant.

```
molecule = Molecule()        # make a new molecule

mu = np.array([[[0, 1, 2],        # define mu's
                [1, 2, 3],
                [2, 3, 4]],

               [[0, 1, 2],
                [1, 2, 3],
                [2, 3, 4]],

               [[0, 1, 2],
                [1, 2, 3],
                [2, 3, 4]]])

molecule.setMu(mu)                # set molecule Mu matrix to mu

molecule.addOmega([1.5,2.5])        # add transition energies
```

```
                                        # to molecule
                                        # (add either individually, or
                                        # as a list)

feyn.linkMolecule(molecule)         # link molecule and feyn objects
```

Now we can calculate the nonlinear response of an individual Feynman object.

```
a_chi = feyn.getChi()         # individual response
```

Or, we can calculate the total nonlinearity for the given system by adding the contributions from each possible Feynman diagram using this function

```
a_chi = feyn.getTotalChi()         # from all permutations
```

Another useful feature is the ability to define a photon object as a *spectrum* photon, whereas previously photons were only defined as having a fixed frequency. A spectrum photon as an input into a Feynman diagram allows for the calculation of a spectrum of χ values, i.e. a χ vs. ω plot, for example.

Following the examples laid out previously, but changing the definition of photon1 to the following:

```
# defining spectrum frequency photons
photon1 = Photon(1,0,spectrum=True,stepfreq=0.01,startfreq=0.0,endfreq=10.0)
```

Following the previous steps aside from this minor deviation, the individual (and total) nonlinearity can also be calculated in an analogous way. Note that the *getChi()* method is versatile enough to function with either a spectrum or fixed frequency photon, except that now two results are returned. (Note: Only define a single photon as a spectrum photon.) The results of this calculation are shown in Fig. 4. Currently, there is nothing to prevent the code from inadvertently dividing by zero. This issue needs to be addressed in the future.

```
wlist,chi = feyn.getChi()         # wlist, chi, are the spectrum and chi, respectively
showChiSpectrum(wlist,chi)         # plotting command
```

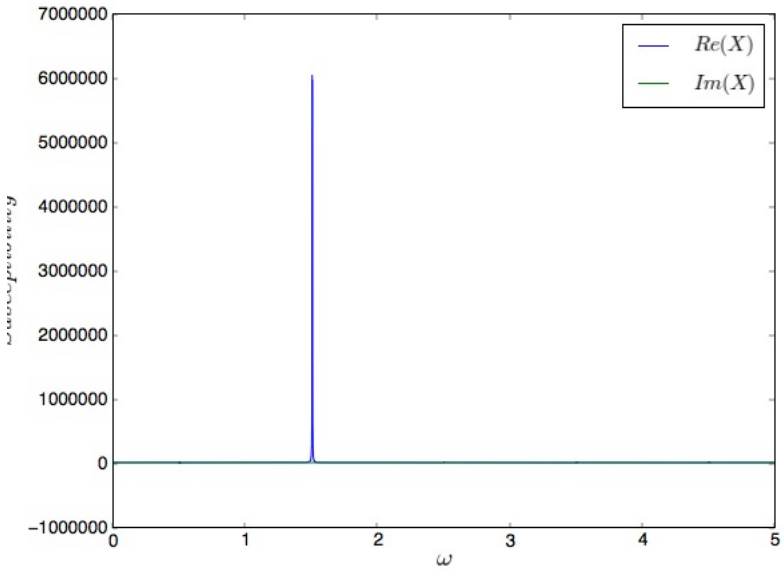

Figure 4: Example $\chi^{(2)}$ spectrum for the aforementioned nonlinear process.

The same can be done for the permutations of the Feynman diagram

```
wlist,chi = feyn.getTotalChi()         # wlist, chi, are the spectrum and chi, respectively
showChiSpectrum(wlist,chi)         # plotting command
```

which in this case results in the same spectrum as shown in Fig. 4.

4.6 Cascading between multiple Feynman diagrams

The bread and butter of FeyPy is its ability to represent nonlinear cascading processes, calculate individual or total nonlinear responses, and return LaTeX strings corresponding to $\chi^{(n)}$, etc. To cascade two diagrams, a second Feynman object must first be defined, in addition to an object of class Cascade. Both Feynman objects must then be added to the Cascade object, which actually handles cascading processes.

```
feyn2 = Feynman()         # new Feynman object
photon4 = Photon(1,0)        # new photons for feyn2
photon5 = Photon(1,0)
photon6 = Photon(-2,0)

feyn2.addPhoton(photon4)         # add new photons to feyn2
feyn2.addPhoton(photon5)
feyn2.addPhoton(photon6)

photon5.toggleVirtual()                # set photon5 to be a virtual

feyn2.linkMolecule(molecule)        # link to first molecule
                                              # for convenience
```

Feel free to check the appearance of *feyn2*. It should be appear to be identical to our first Feynman diagram. The two diagrams can be cascaded as follows:

```
cascade = Cascade()         # create Cascade object

cascade.addFeynman(feyn)         # add feynman diagrams
cascade.addFeynman(feyn2)         # to cascade

cascade.showDiagram()         # show cascaded diagram
```

This results in the diagram shown in Fig. 5.

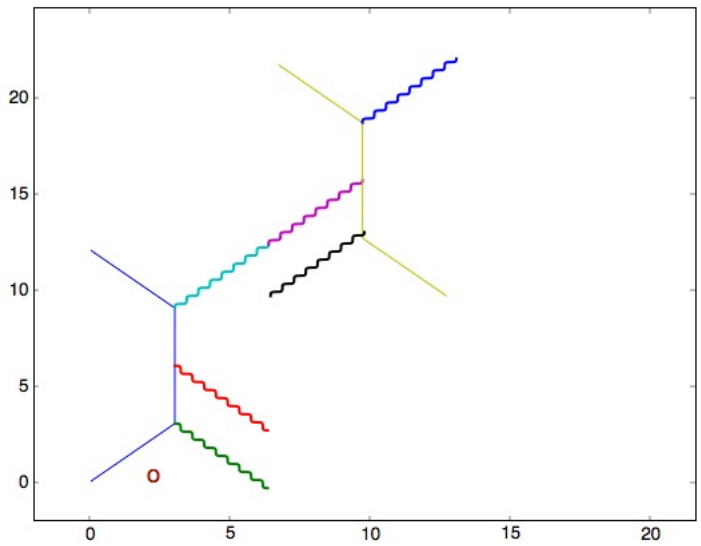

Figure 5: Example of a cascading process.

5 Known Issues

Have you had a problem with the code? Did it give output that appears wonky or inconsistent? Did you receive a strange error that you don't understand? Did your computer explode? Let us know, particularly (1) what you were trying to do (i.e. *exactly* what you did), (2) what you expected to happen, and (3) what actually happened. Bonus points for screenshots or verbatim text of the error you receive, in addition to system specifications and how you are running the code. Please send all bugs to the email listed below the title of this document, adamgoler@email.wsu.edu. Thanks!

6 Feature Request

Do you have an idea to improve FeyPy? By all means, we'd love to hear it! Please send all requests to the email listed below the title of this document, adamgoler@email.wsu.edu. Thanks!

- If one of the molecules emits two photons, is there a way to assign the one of choice for cascading? I assume you could just use photonX.toggleVirtual() for the output photon too. Also, what if there are two or more virtual photons exchanged? -Kuzyk

7 Code Glossary

Continually updating...