Contents

About

What Is IOpy?

IOpy is a python-based package for solving the equations of motion of coupled oscillators which are in contact with thermal baths. These equations which are in the form of Langevin equations (or quantum Langevin equations in quantum limit) appear in many research areas in physics. Specifically, in optomechanics, which is the study of interaction of light with mechanical oscillators, this problem forms the essence of the theory. Finding the solutions of Langevin equations in different optomechanical systems can help with discovering and understanding

novel phenomena. However, the procedure of finding these solutions is similar in many cases (input-output formalism, which is where the name IOpy is coming from) and the essential difference between them is the difference between the physical setups. Moreover, in complex setups the calculations necessary to find the solutions can be hard and tedious to do by hand (for example inverting matrices with large dimensions).

On the other hand, in many problems in optomechanics there are a lot of physical phenomena which are involved in the dynamics. For newcomers to the field, like students who want to learn optomechanics, it can be confusing to distinguish between the different effects involved in the dynamics. Looking for a solution to resolve the two mentioned issues was the motivation to write this code.

With IOpy, you can define your physical setup very fast and without the need for going through the details. Further more you can visualize the results in a way which can help people to test their theoretical results and also help newcomers to grasp the elements of the optomechanics. For example to see the emission spectrum of a hot optical resonator you can define your optical mode in a single line:

```
a = Mode(name = 'a', omega = 5e9 *2*np.pi)
```

And then defining the thermal bath and the driving field can be done each in a single line:

```
a_inex = Input(name = 'ex', a, kappa = 0.2e6 *2*np.pi,
               kind = 'drive', omega_drive = 5e9 *2*np.pi,
               bath_temp = 2e-5)
a_in0 = Input('0', a, kappa = 0.3e6 *2*np.pi, kind = 'bath',
              bath_temp = 10e-3)
```

And finally defining the system, the output port and the spectrum:

```
sys_cav = System([a], [a_in0,  a_inex], [])
a_outex = Output(sys_cav, a_inex)
spec = me.spectrum(omegas, me.PowerMeasurement(a_outex),
                   components = False, plot = True)
```

And the result would be:

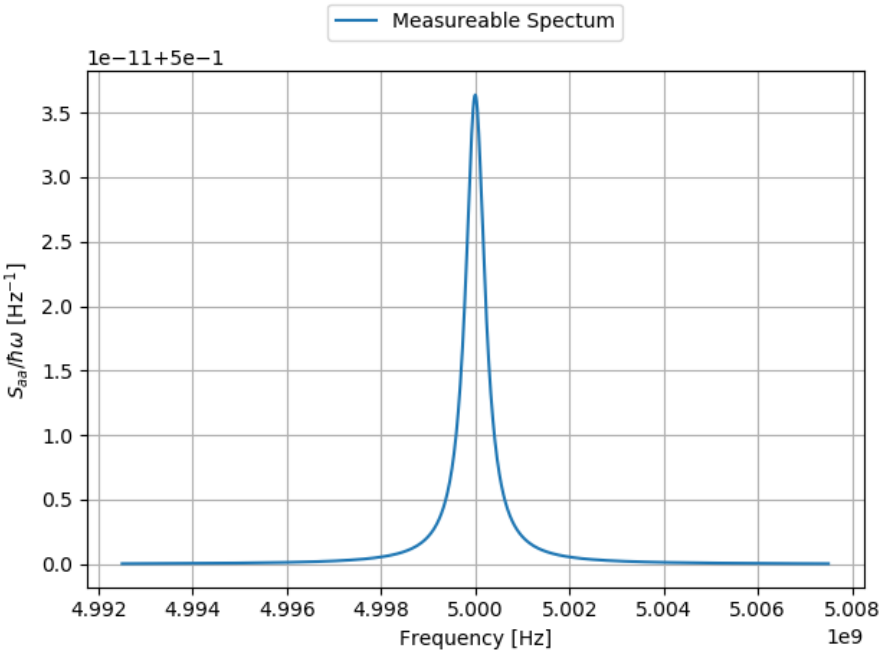

Figure 1: Simple cavity output spectrum

A more detailed explanation of this example as well as more examples with respect to optomechanics are available on the Examples page.

Structure of IOpy

IOpy consists out of four scripts which each serving a special purpose:

`elements`: For defining different components of the physical system (modes, couplings, input-output fields and the system)

`DCnonlinearities`: For calculation of DC shifts in the system variables due to nonlinear effects.

`measurement`: For calculating linear responses and power spectral densities.

`plots`: For visualizing the measurements' results in graphs.

Installation

To install IOpy you have to clone the IOpy repository on your local computer. The packages you need for using IOpy are `numpy`, `scipy` and `matplotlib`.

Theory

Input-Output Formalism

On this Page, the theory behind the IOpy calculations is introduced.

There are different approaches to model open quantum systems. But the two most commonly used ones are the density matrix approach using master equations and the Langevin equations, which is also known as input-output theory. Input-output theory allows us to directly model the quantum fluctuations injected from any port into the system. Quantum Langevin equations are formulated on the level of the Heisenberg equations of motion describing the time evolution of any operator of the system. Moreover, in case of optical cavities any coherent laser drive that may be present can be taken into account. For more details see Gardiner and Zoller (2004) and Clerk et al. (2010).

For example, for an optical cavity which is driven by a coherent laser field with the detuning Δ from the cavity resonance frequency, via a coupling of κ_{ex} and is also coupled to the environment through the dissipation rate of κ_0, the equations of motion can be written as

$$\dot{a} = -\frac{\kappa}{2}a + i\Delta a + \sqrt{\kappa_{\mathrm{ex}}}a_{\mathrm{in,ex}} + \sqrt{\kappa_0}a_{\mathrm{in},0},$$

where $\kappa = \kappa_0 + \kappa_{\mathrm{ex}}$ is the total dissipation rate. $a_{\mathrm{in,ex}}$ and $a_{\mathrm{in},0}$ are input operators of the driving field and the stochastic thermal field respectively. This equation can also be written in terms of quadrature operators

$$\dot{X} = \Delta Y - \frac{\kappa}{2}X + \sqrt{\kappa_{\mathrm{ex}}}\,X_{\mathrm{in,ex}} + \sqrt{\kappa_0}\,X_{\mathrm{in},0}$$
$$\dot{Y} = -\Delta X - \frac{\kappa}{2}Y + \sqrt{\kappa_{\mathrm{ex}}}\,Y_{\mathrm{in,ex}} + \sqrt{\kappa_0}\,Y_{\mathrm{in},0},$$

where the quadrature operators are defined as

$$X = \frac{a^\dagger + a}{\sqrt{2}},$$
$$Y = i\frac{a^\dagger - a}{\sqrt{2}}.$$

According to the input-output theory of open quantum systems, the output field (which in case of optical cavity can be the reflected or transmitted light through the cavity) is given by

$$a_{\mathrm{out}} = a_{\mathrm{in}} - \sqrt{\kappa_{\mathrm{ex}}}a.$$

It can also be written in terms of quadratures

$$X_{\mathrm{out,ex}} = X_{\mathrm{in,ex}} - \sqrt{\kappa_{\mathrm{ex}}}\, X,$$

$$Y_{\mathrm{out,ex}} = Y_{\mathrm{in,ex}} - \sqrt{\kappa_{\mathrm{ex}}}\, Y.$$

Any set of Langevin equations, linearized around a stationary solution, can in principle be written in the form

$$\dot{Z} = \mathbf{M}Z + \mathbf{L}Z_{\mathrm{in}}, \tag{1}$$

with the input-output relations

$$Z_{\mathrm{out}} = Z_{\mathrm{in}} - \mathbf{L}^T Z,$$

where Z is a vector containing the quadrature operators of different oscillators involved in the dynamics. Z_{in} and Z_{out} are vectors containing the quadrature operators of the input and output fields. For the optical cavity of the example above we have

$$Z = \begin{pmatrix} X \\ Y \end{pmatrix}, \quad Z_{\mathrm{in}} = \begin{pmatrix} X_{\mathrm{in,ex}} \\ Y_{\mathrm{in,ex}} \\ X_{\mathrm{in,0}} \\ Y_{\mathrm{in,0}} \end{pmatrix}, \quad Z_{\mathrm{out}} = \begin{pmatrix} X_{\mathrm{out,ex}} \\ Y_{\mathrm{out,ex}} \\ X_{\mathrm{out,0}} \\ Y_{\mathrm{out,0}} \end{pmatrix}$$

and

$$\mathbf{M} = \begin{pmatrix} -\frac{\kappa}{2} & \Delta \\ -\Delta & -\frac{\kappa}{2} \end{pmatrix}, \quad \mathbf{L} = \begin{pmatrix} \sqrt{\kappa_{\mathrm{ex}}} & 0 & \sqrt{\kappa_0} & 0 \\ 0 & \sqrt{\kappa_{\mathrm{ex}}} & 0 & \sqrt{\kappa_0} \end{pmatrix}.$$

Due to the linearity of the equations one can take the Fourier transform of the equations and define a scattering matrix which relates the output fields to the input fields

$$Z_{\mathrm{out}}(\omega) = \mathbf{S}(\omega) Z_{\mathrm{in}}(\omega), \tag{2}$$

where

$$\mathbf{S} = 1 + \mathbf{L}^T (i\omega + \mathbf{M})^{-1} \mathbf{L}. \tag{3}$$

Measurements

IOpy implements two different methods to characterize a linear system. They mimic closely the different types of measurements performed in the laboratory. First is the linear response of the system, which is similar to output of a vector network analyser (VNA). And, second is the spectrum of the output field, which is the same as the output of the spectrum analyser.

Linear Response

According to the Equation (2), a quadrature pair of any output port i can be related to the quadrature pair of any input port j using a submatrix of the scattering matrix

$$\begin{pmatrix} X_{\mathrm{out},i} \\ Y_{\mathrm{out},i} \end{pmatrix} = \mathbf{S}^{(ij)} \begin{pmatrix} X_{\mathrm{in},j} \\ Y_{\mathrm{in},j} \end{pmatrix}.$$

Then the output field operator can be written as

$$a_{\mathrm{out},i} = \frac{X_{\mathrm{out},i} + iY_{\mathrm{out},i}}{\sqrt{2}} = (\mathbf{S}_{11}^{(ij)} + i\mathbf{S}_{21}^{(ij)})\frac{X_{\mathrm{in},j}}{\sqrt{2}} + (\mathbf{S}_{22}^{(ij)} - i\mathbf{S}_{12}^{(ij)})\frac{iY_{\mathrm{in},j}}{\sqrt{2}}.$$

In many cases the input field is a coherent drive with constant amplitude and $X_{\mathrm{in},j}$ and $Y_{\mathrm{in},j}$ are cosine part and sine part of it. Therefore, choosing any arbitrary values for the quadratures, as long as they satisfy the identity $X_{\mathrm{in},j}^2 + Y_{\mathrm{in},j}^2 = 1$, will only impose a total phase shift to the response. By choosing $Y_{\mathrm{in},j}$ to be zero, we consider this phase shift to be zero and $a_{\mathrm{in},j} = X_{\mathrm{in},j}/\sqrt{2}$. as a result, any phase response in the output is being calculated with respect to the input field.

$$a_{\mathrm{out},i} = (\mathbf{S}_{11}^{(ij)} + i\mathbf{S}_{21}^{(ij)})a_{\mathrm{in},j}$$

or in a more familiar notation

$$\chi^{(ij)}(\omega) = \mathbf{S}_{11}^{(ij)} + i\mathbf{S}_{21}^{(ij)}. \tag{4}$$

Spectra

In general for analyzing the spectrum of some measured signals, we can define a correlator using a measurement matrix Q_{ij}

$$Q(\tau) = \langle Q_{ij} Z_{\mathrm{out},i}(0) Z_{\mathrm{out},j}(\tau) \rangle,$$

where we have used the Einstein notation for the summations. The Fourier transform of this correlator would be the spectral density or spectrum

$$S_{QQ}[\omega] = \int_{-\infty}^{\infty} Q(\tau) e^{i\omega\tau} d\tau.$$

Due to the linearity of the expectation value and the integral, the spectrum can be rewritten in the following way

$$S_{QQ}[\omega] = \frac{1}{2\pi} Q_{ij} \int_{-\infty}^{\infty} \langle Z_{\mathrm{out},i}(\omega_1) Z_{\mathrm{out},j}(\omega) \rangle d\omega_1.$$

According to Equation (2), we can write the output signals in terms of input signals

$$\begin{aligned} Z_{\mathrm{out},i}(\omega) &= S_{ik}(\omega) Z_{\mathrm{in},k}(\omega), \\ S_{QQ}[\omega] &= \frac{1}{2\pi} Q_{ij} \int_{-\infty}^{\infty} S_{ik}(\omega_1) S_{jl}(\omega) \langle Z_{\mathrm{in},k}(\omega_1) Z_{\mathrm{in},l}(\omega) \rangle d\omega_1. \end{aligned} \tag{5}$$

In general, the input signals can be correlated with each other and have complicated statistical behaviors. In many situations (such as thermal input noises) it's a good approximation to consider different sources to be uncorrelated. More precisely

$$\langle Z_{\mathrm{in},k}(\omega_1) Z_{\mathrm{in},l}(\omega) \rangle = 2\pi \delta_{kl} \mathcal{S}_k^{\mathrm{in}}(\omega) \delta(\omega_1 + \omega),$$

where $\mathcal{S}_k^{\mathrm{in}}(\omega)$ is the spectral density of the k^{th} input signal. Now, we can simplify Equation (5) to

$$S_{QQ}[\omega] = Q_{ij} S_{ik}(-\omega) S_{jk}(\omega) \mathcal{S}_k^{\mathrm{in}}(\omega).$$

This final result is used in the calculations of IOpy. Here we make a remark on the summations by rearranging the terms in the following way

$$S_{QQ}[\omega] = \sum_k \mathcal{S}_k^{\mathrm{in}}(\omega) \Big(\sum_{i,j} Q_{ij} S_{ik}(-\omega) S_{jk}(\omega) \Big) = \sum_k c_k(\omega) \mathcal{S}_k^{\mathrm{in}}(\omega).$$

This illustrates that the spectrum can be expressed as a sum of the different input noise contributions.

Library

In this section the different classes and functions of IOpy are introduced. IOpy consists of 4 main scripts `elements`, `measurement`, `DCnonlinearaity` and `plots`.

elements

Objects and functions in this script are used to define the oscillating modes, input-output ports, couplings between different modes and finally the whole system of coupled oscillators.

Class `Mode`

This class represents the harmonic oscillators modes.

```
'''
Attributes:
    name: name of the mode.
    omega: resonance frequency of the mode in rad/sec.
    kappa: mode total dissipation rate in rad/sec.
    omega_rot: frequency at which the mode frame is
               rotating in rad/sec.
    driven: flag indicates whether the mode is driven or not.

Properties:
    omega_d():
       returns the frequency of the field which is driving
       the mode in rad/sec.
'''
```

Class `Input`

This class represents the input field coupled to a `Mode`. Inputs can be coherent drives (pumps) or thermal baths where the only difference between them is the rotating frame.

```
'''
Attributes:
    name: name of the input field.
    mode: the mode which the input is coupled to.
    kind: flag which indicates the input is a pump or a thermal
          bath. 'drive' for a pump and 'bath' for a thermal bath.
    kappa: coupling rate to the mode in rad/sec.
    omega_drive: frequency of the pump fields in rad/sec.
    bath_temp: temperature of the bath or the pump in Kelvins.
```

```
    nbar: average number of thermal photons in the input field.
'''
```

`nbar` is calculated using the formula

$$\bar{n} = \frac{1}{e^{\frac{\hbar\omega}{kT}} - 1}.$$

Methods:

```
'''
spectrum(self, omegas):
    spectrum of the input field. Here we approximately take
    the thermal spectrum to be constant near the mode frequency.
    Args:
        omegas: the frequencies vector at which we want to
                calculate the spectrum.

    Returns:
            spectrum of the input field in units of number
            of photons.
'''
```

Currently for the input noise spectrum we consider the simplest case that it is a thermal spectrum. In future a feature will be added that allows for user defined spectra. This would make it possible to account for the classical noises of the lasers (phase noise and amplitude noise).

Class `Coupling`

This class represents the couplings between two `Mode`s using the coupling vector. The coupling vector V_g is a 4-dimensional vector which is defined in a way that the interaction Hamiltonian for two coupled modes would be

$$H_{\text{int}} = 2\hbar(V_{g,1}X_1X_2 + V_{g,1}X_1Y_2 + V_{g,1}Y_1X_2 + V_{g,1}Y_1Y_2).$$

For optomechanics the couling vector is

$$V_g = (g, 0, 0, 0).$$

```
'''
Attributes:
    mode1: first mode.
    mode2: second mode.
    vg: coupling vector. a 4-d real vector.
'''
```

Methods:

```
'''
contains_mode(self, mode):
    indicates if the mode is involved in this coupling or not.

    Args:
        mode: the mode we want to look for.

    Returns:
       'True' if the mode is involved and 'False' for otherwise.
'''
```

Class `System`

Defines the complex system made of coupled `Modes` and `Inputs`. Its most important purpose is to calculate the scattering matrix.

```
'''
Attributes:
    modes: an array of modes in the system.
    inputs: an array of inputs of the system.
    couplings: couplings between the modes of the system.
    M: the M matrix in the relation dZ/dt = M*Z + L*Z_in.
    L: the L matrix in the relation dZ/dt = M*Z + L*Z_in.
'''
```

Refer to Equation (1) on the Theory page for more information on M and L matrices.

Methods:

```
'''
add_mode(self, mode):
    Adding a mode to the system.

    Args:
        mode: the mode we want to add.
'''

'''
add_input(self, inp):
    Adding an input to the system.

    Args:
        inp: the input we want to add.
'''
```

```
'''
add_coupling(self, coup):
    Adding a coupling to the system.

    Args:
        coup: the coupling we want to add.
'''

'''
make_ML(self):
    Constructing M and L matrices of the system.
'''

'''
SMatrix(self, omegas):
    Constructs the scattering matrix of the system for a
    frequency range.

    Args:
        omegas: frequencies vector in rad/sec.

    Returns:
        Ss: the scattering matrix.
'''
```

Refer to Equation (3) on Theory page for more information about scattering matrix.

Class `Output`

This class represents the output field of the system with respect to an input field (in terms of input-output formalism).

```
'''
Attributes:
    system: the complex system of coupled modes and inputs.
    input: the input field which we want to define its
           output field (in terms of input-output formalism).
    mode: the mode which these input and output fields are coupled.
'''
```

measurement

Objects and functions in this script are used to perfom measurements on the output fields. The possible measurement schemes are linear response and spectrum. For more information on the definitions see Paragraph measurements on the theory Page.

Class `MeasurementOperator`

This class represents a general measurement with a measurement matrix for defining a correlator

$$Q(\tau) = \langle Q_{ij} Z_{\text{out},i}(0) Z_{\text{out},j}(\tau) \rangle,$$

where Q_{ij} is the ij^{th} element of the measurement matrix.

Refer to the Paragraph spectra of the theory Page for more information on the measurement matrix.

```
'''
Attributes:
    Q: the measurement matrix.
    system: the system which the output field is coming from.
    omega_d: the driving frequency of the mode which the
             output field is coming from.
'''
```

Class `PowerMeasurement`

This class represents a power measurement scheme object. The correlator function and measurement matrix in this scheme are

$$Q(\tau) = \langle X(0)X(\tau) + iX(0)Y(\tau) - iY(0)X(\tau) + Y(0)Y(\tau) \rangle,$$

$$[Q] = \begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix}.$$

```
'''
Attributes:
    system: the system which the output field is coming from.
    omega_d: the driving frequency of the mode which the
             output field is coming from.
    Q: the measurement matrix.
'''
```

Class `HomodynMeasurement`

This class represents a Homodyn measurement scheme object with a homodyning angle. The correlator function and measurement matrix in this scheme are

$$\begin{aligned} Q(\tau) = \langle &\cos^2(\theta) X(0)X(\tau) + \sin(\theta)\cos(\theta) X(0)Y(\tau) \\ &+ \sin(\theta)\cos(\theta) Y(0)X(\tau) + \sin^2(\theta) Y(0)Y(\tau) \rangle, \end{aligned}$$

$$[Q] = \begin{pmatrix} \cos^2(\theta) & \sin(\theta)\cos(\theta) \\ \sin(\theta)\cos(\theta) & \sin^2(\theta) \end{pmatrix}.$$

```
'''
Attributes:
    system: the system which the output field is coming from.
    omega_d: the driving frequency of the mode which the
             output field is coming from.
    Q: the measurement matrix.
'''
```

Function `linear_response`

This function calculates linear response (susceptibility) of the system from one specific input port to an output port in frequency domain

$$a_{\text{out}} = \chi a_{\text{in}}.$$

Refer to Equation (4) of linear response Paragraph on the theory Page for more information on linear response.

```
'''
    Args:
        Omegas: the frequencies vector we want to calculate
                the linear response for them, in frame of the
                input field (not a rotating frame)
        system: the system which we want to measure its response.
        output: the output port.
        Input: the input port.
        plot: flag, indicates to plot the susceptibilities or not.

    Returns:
        omegas_out: the frequencies vector we want to calculate
                    the linear response for them, in frame of the
                    output field (not a rotating frame)
        chi: the susceptibility we want to measure.
'''
```

Function `spectrum`

The spectrum of an output field. Refer to spectra Paragraph on the theory Page for more information on the spectra.

```
'''
    Args:
        omegas: the frequencies vector we want to calculate the
                spectrum for them, in frame of the output field
```

```
                (not a rotating frame)
            measurement: the measurement scheme, of kinds PowerMeasurement
                         or HomodynMeasurement or another general measurement.
            components: flag, indicates to calcuate different contributions
                        of noise sources or just calculate the whole spectrum.
            plot: flag, indicates to plot the spectra or not.

        Returns:
            spec: the spectrum of the output field in case of components = False.
                  A list of spectra from different contributions in case of
                  components = True. The nth element of the list contains the
                  comulative sum of first n contributions.
    '''
```

DCnonlinearities

Functions in this script are used to calculate the DC shifts resulting from nonlinear effects.

Function `Kerr_effect_nbar`

This function finds the steady state average photon number in an optical cavity with kerr type nonlinearity. It findes the smallest root of a third order polynomial equation:

$$[\frac{-\kappa_{\mathrm{ex}} P_{\mathrm{in}}}{\hbar\omega_{\mathrm{drive}}}]\bar{n}^3 + (\Delta^2 + (\frac{\kappa}{2})^2)\bar{n}^2 + (2K\Delta)\bar{n} + (K^2) = 0$$

```
    '''
        Args:
            P_in: input power in Watts.
            kappa_0: cavity intrinsic dissipation rate in rad/sec.
            kappa_ex: input coupling rate in rad/sec.
            omega_c: cavity resonance frequency in rad/sec.
            omega_drive: frequency of the input field in rad/sec.
            K: nonlinearity coefficient in rad/sec.

        returns:
            smallest real root of the third order polynomial equation.
    '''
```

Function `optomechanics`

This function finds the steady state average photon number in an optomechanical cavity and also finds the DC shift cavity resonance frequency. It uses the `Kerr_effect_nbar()` function to solve the third order equation

$$\bar{n}[\frac{\kappa^2}{4} + (\Delta - (\frac{2g_0^2}{\Omega_m})\bar{n})^2] = \kappa_{\text{ext}}\frac{P_{\text{in}}}{\hbar\omega_{\text{drive}}}.$$

```
'''
    Args:
        P_in: input power in Watts.
        kappa_0: cavity intrinsic dissipation rate in rad/sec.
        kappa_ex: input coupling rate in rad/sec.
        omega_c: cavity resonance frequency in rad/sec.
        omega_drive: frequency of the input field in rad/sec.
        omega_m: resonance frequency of the mechanical oscillator in rad/sec.
        g_0:  vacuum optomechanical coupling rate in rad/sec.

    returns:
        omega_c: modified cavity resonance frequency in rad/sec.
        g: optomechanical coupling rate in rad/sec.
'''
```

plots

Functions in this script can be used for plotting the linear responses and spectra.

Function `plot_linear_response`

This function is used for plotting the linear response functions. It plots the absolute value and phase of the linear response as well as the linear response as a trajectory in the complex plane.

```
'''
        Args:
            omegas: the vector containing the frequencies
                    in rad/sec (not in a rotating frame).
            chi: the linear response function.
            system: the system which the linear response is from.
            output: the output field we that the linear response
                    is calculated for.
            input: the input field we that the linear response
                   is calculated for.
'''
```

Function `plot_spectrum`

This function is used for plotting the spectra.

```
'''
        Args:
            omegas: the vector containing the frequencies
                    in rad/sec (not in a rotating frame).
            spec: the spectrum.
            componenets: flag, indicates to plot different contributions
                         of noise sources in the spectrum or just
                         plot the whole spectrum.
            system: the system which the spectrum is from.
'''
```

Examples

In this section a set of famous phenomena in optomechanics is presented. These calculations serve first as examples of using IOpy in simulating Langevin equations, and second are benchmarks of IOpy by comparing it to known theoretical models.

Simple Cavity

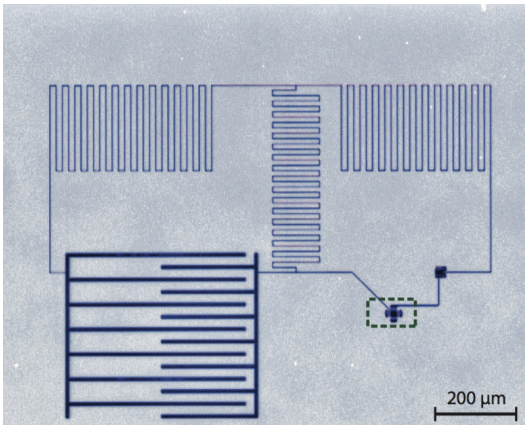

Figure 2: Inverted-colour optical micrograph of the circuit consisting of two coupled LC resonators, one having a mechanically com- pliant capacitor (Toth et. al. 2017).

in the first example we illustrate how to simulate a hot microwave resonator. Here the temperature of the bath is higher than the temperature of the drive

and therefore we expect to see an emission shaped like a Lorentzian.

Let's start by defining the cavity mode

```
omega_c = 5e9*np.pi*2
a = Mode('a', omega_c)
```

Then we have to define input fields. In this example the cavity is driven by a coherent drive and it is coupled to a thermal bath.

```
kappa_ex = 0.2e6*np.pi*2
kappa_0 = 0.3e6*np.pi*2
kappa = kappa_ex + kappa_0

T_drive = 2e-5
T_bath = 10e-3

a_inex = Input('ex', a, kappa_ex, kind = 'drive',
                omega_drive = omega_c, bath_temp=T_drive)
a_in0 = Input('0', a, kappa_0, kind = 'bath', bath_temp=T_bath)
```

And finally we define the system object and the output field.

```
sys_cav = System([a], [a_in0,  a_inex], [])

a_outex = Output(sys_cav, a_inex)
```

Now for measuring the spectrum of the output field, we have to use the `spectrum` function.

```
omegas = np.linspace(omega_c - 15*kappa, omega_c + 15*kappa, 1001)
spec = me.spectrum(omegas, me.PowerMeasurement(a_outex),
                   components = False, plot = True)
```

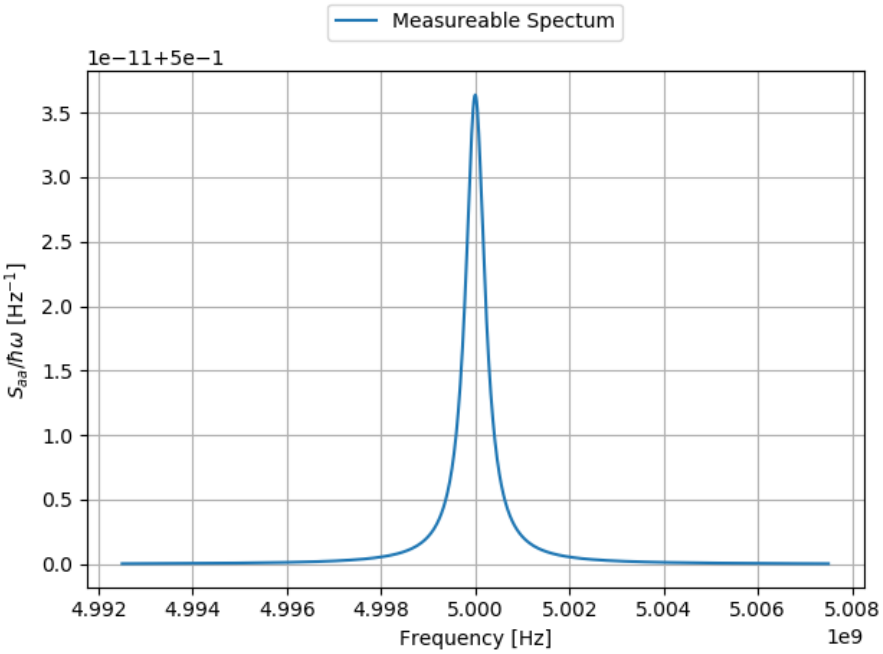

Figure 3: Simple cavity output spectrum

As we expect, we can see a Lorentzian in the spectrum. If the temperature of the driving field was higher than the thermal bath, we would see a deep instead of a peak.

The linear response of the system to driving can also be measured with the `linear_response` function

```
omegas_newex, S_ex = me.linear_response(omegas, sys_cav, a_outex,
                                        a_inex, plot = 1)
```

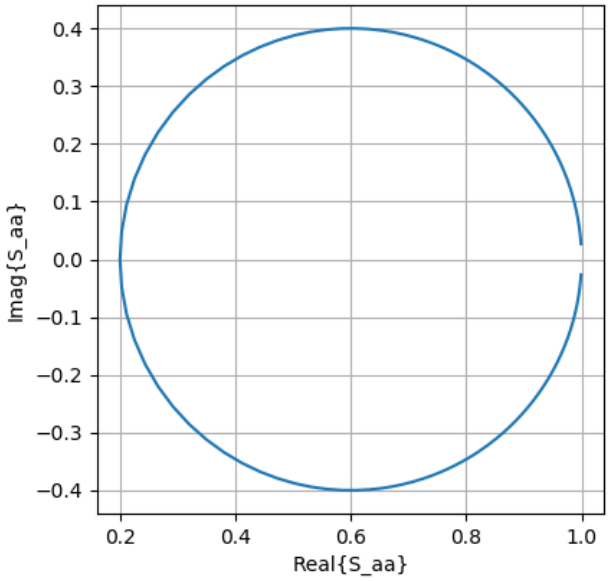

Figure 4: Cavity linear response in complex space

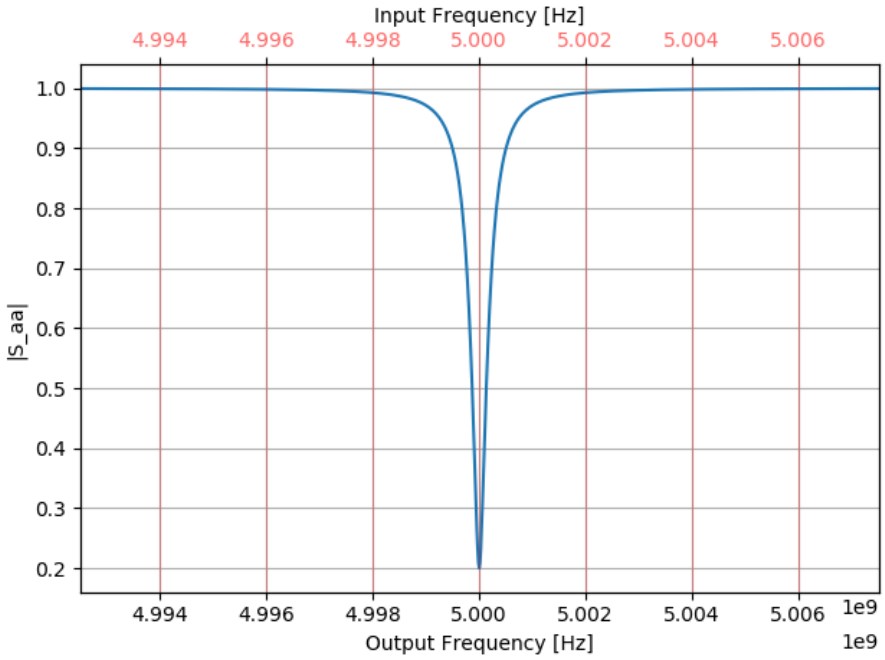

Figure 5: Cavity linear response amplitude

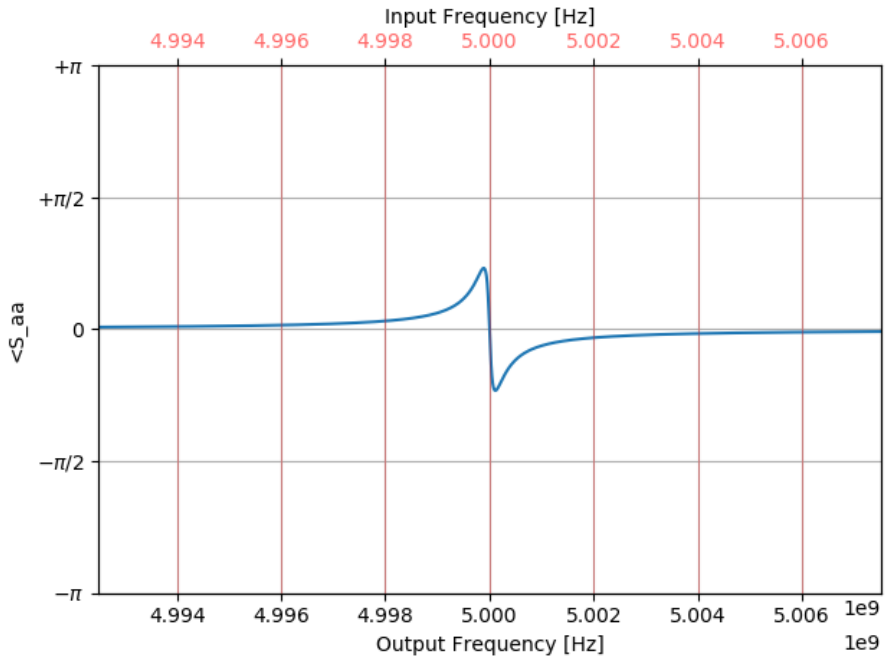

Figure 6: Cavity linear response phase

The results can also be compared to the theory. For example in this case the linear response of the system to the drive field is

$$S_{aa} = 1 - \frac{\kappa_{ex}}{\frac{\kappa}{2} - i(\omega - \omega_c)}.$$

The graphs below show the comparison between this equation and IOpy results.

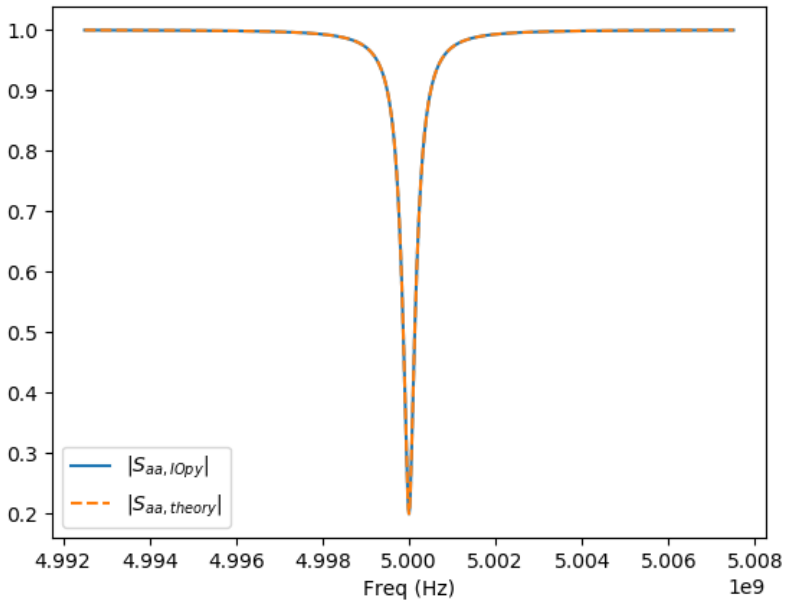

Figure 7: Cvity linear response amplitude calculated by theory (dashed) and IOpy (blue)

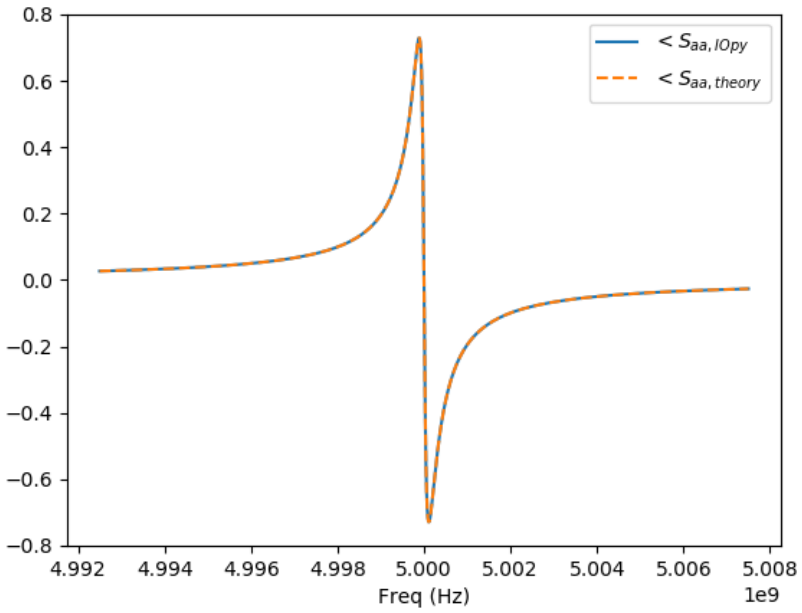

Figure 8: Cvity linear response phase calculated by theory (dashed) and IOpy (blue)

Basic Optomechanics and Cooling

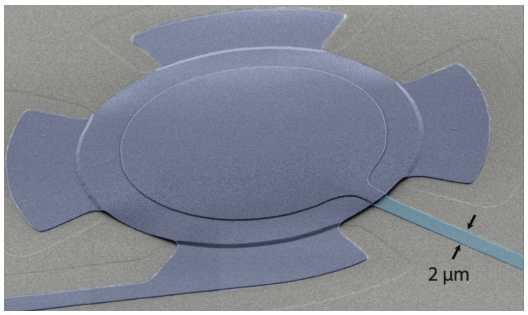

Figure 9: False-colour scanning electron micrograph of the mechanically compliant drum capacitor (Toth et. al. 2017).

In this example we simulate an optomechanical system with a weak drive. Here we want to see the optomechanical cooling due to the increase in optomechanical

damping rate.

In the basic optomechanical interaction, the cavity resonance frequency shifts by a constant value due to the DC nonlinearity. Therefore, before defining the modes we have to calculate this DC shift.

```
omega_c = 5e9*np.pi*2     # cavity resonance frequency

kappa_0 = 0.3e6*np.pi*2
kappa_ex = 0.4e6*np.pi*2

kappa = kappa_0 + kappa_ex

omega_m = 5e6*np.pi*2    # mechanical resonance frequency
gamma_m = 100*np.pi*2

P_in = 5e-12

g_0 = 200*np.pi*2

omega_drive = omega_c - 1* omega_m


from DCnonlinearities import optomechanics

omdir = optomechanics(P_in, kappa_0, kappa_ex, omega_c,
                      omega_drive, omega_m, g_0)

g= omdir['g']  # optomechanical coupling rate = sqrt(nbar)*g_0
omega_c = omdir['omega_c']   # new cavity resonance frequency
```

Now we can define the modes, as well as input fields including thermal baths coupled to optics and mechanics and the optical driving field.

```
a = Mode('a', omega_c)
b = Mode('b', omega_m)

a_inex = Input('ex', a, kappa_ex, kind = 'drive',
                omega_drive = omega_drive, bath_temp=10e-3)
a_in0 = Input('0', a, kappa_0, kind = 'bath', bath_temp=10e-3)

b_in0 = Input('0', b, gamma_m, kind = 'bath', bath_temp=10e-3)
```

Then we should define the coupling between the optical and mechanical modes:

```
g_ab = Coupling(a, b, g * np.array([1,0,0,0]))
```

And finally we define the whole optomechanical system:

```
sys_om = System([a, b], [a_in0,b_in0 , a_inex], [g_ab])
```

Now just like the previous example we can measure the spectrum of the output field.

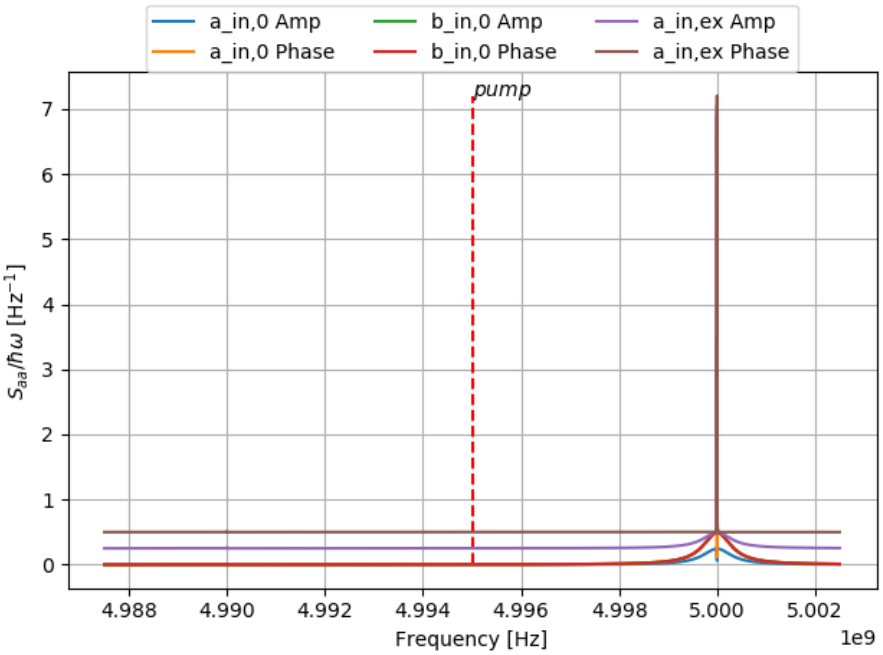

Figure 10: Optomechanical cavity output spectrum

The peak that can be seen is because of low sampling rate of the calculations. To have a better precision we zoom on the neighborhood of the cavity resonance frequency.

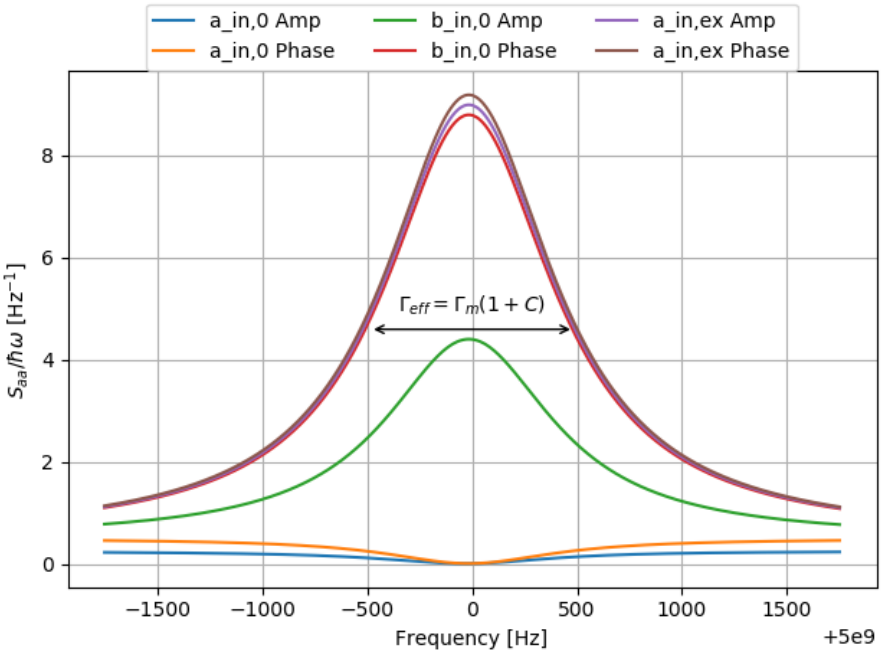

Figure 11: Optomechanical cavity output spectrum

Here we can clearly see different contributions to the spectrum. In addition the width of the spectrum is equal to the theory value $\Gamma_{\text{eff}} = \Gamma_m(1 + C)$ (with C as the cooperativity equal to $\frac{4g^2}{\kappa\Gamma_m}$) which results in cooling.

Strong Coupling Regime

In this example we show the effect of increasing of the laser input power (P_{in}). For the details on the theory see (Aspelmeyer, Kippenberg, Marquardt (2014)), Section VII.C. By increasing the input power, at the beginning we can observe an improvement in the cooling, but as we continue increasing the power, the optical and mechanical modes hybridize to form two new modes with the eigenfrequencies

$$\omega_{\pm} = \frac{\Omega_m - \Delta}{2} \pm \sqrt{g^2 + (\frac{\Omega_m + \Delta}{2})^2}.$$

When the driving laser is exactly detuned on the red sideband ($\Delta = -\Omega_m$) the splitting of these two modes is equal to $2g$. In this example we want to show this splitting on the spectrum.

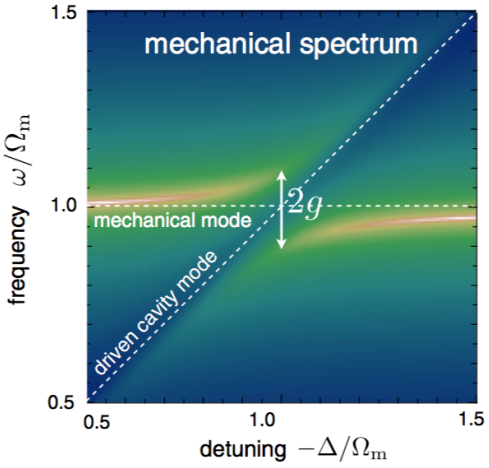

Figure 12: Mechanical frequency spectrum (frequency on vertical axis) as a function of laser detuning, for a strongly coupled optomechanical system (Aspelmeyer, Kippenberg, Marquardt 2014).

To simulate this phenomenon, the code is exactly the same as the previous example but with a different input power.

```
P_in = 5e-9
```

By measuring the output filed spectrum we can clearly see this mode splitting.

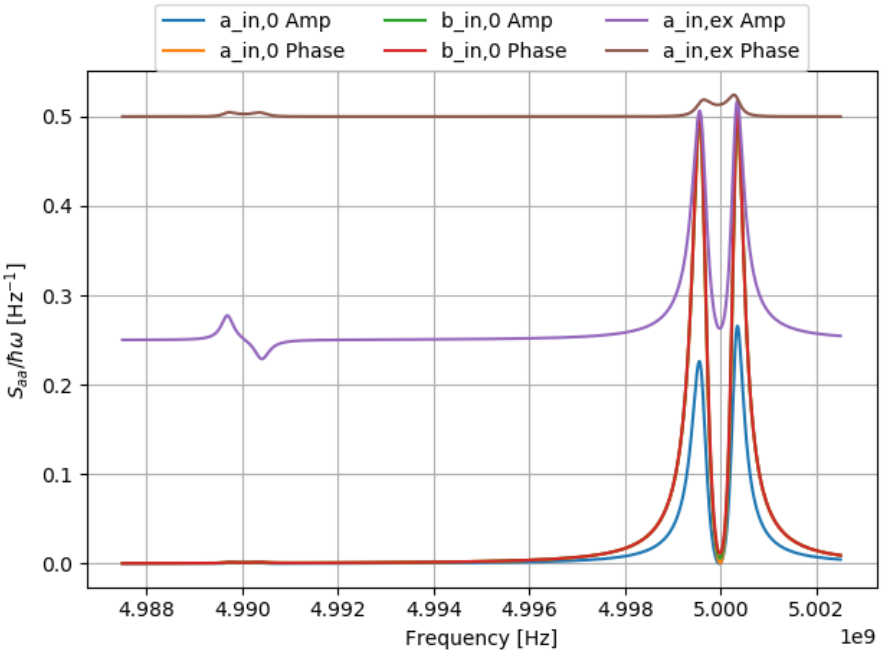

Figure 13: Optomechanical cavity output spectrum. One can see both red and blue sidebands of the pump.

To see better the splitting we change the measurement frequencies.

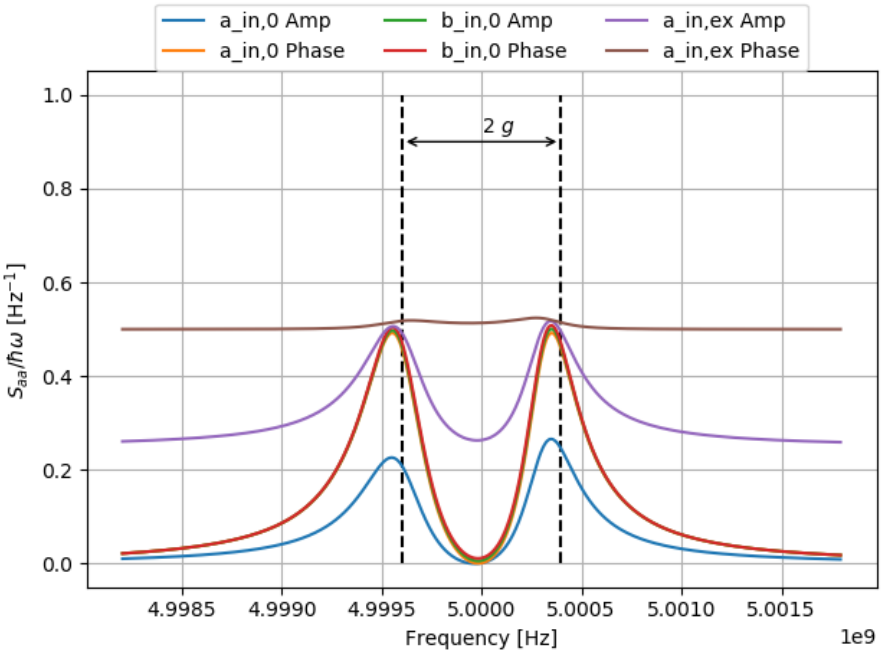

Figure 14: Optomechanical cavity output spectrum

Optomechanically Induced Transparency

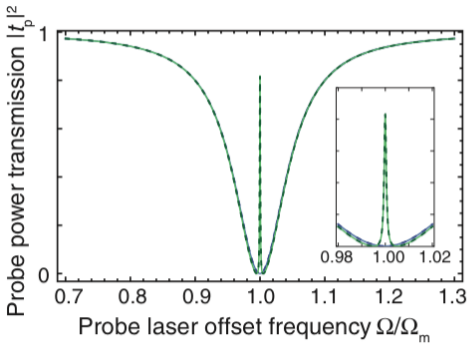

Figure 15: Transmission of the probe laser power through the optomechanical system in the case of a critically coupled cavity as a function of normalized probe laser frequency offset, when the control field is off (blue lines) and on (green lines) (Weis et al., 2010)

This example is about the optomechanically induced transparency effect also known as OMIT. This effect was observed in atoms (electromagnetically induced transparency Fleischhauer, Imamoglu, and Marangos, 2005) as the cancellation of absorption in the presence of an auxiliary laser field. OMIT was predicted theoretically by Schliesser, 2009 and Agarwal and Huang 2010. When the optical cavity is pumped on the red sideband and we inject a weak probe field into the cavity, the optomechanical interaction causes the cavity to be seen transparent by this weak field.

To simulate this phenomenan in IOpy, the setup is again similar to the basic optomechanics example. The difference here is that we set the driving field (control field, in this context) to be a high temperature source. In this way, the noise around this control field play the role of the weak probe field for us, so we can see the OMIT effect in the spectrum, as well as the linear response.

```
T_cont = 1
T_bath = 10e-3

a_cont = Input('ex', a, kappa_ex, kind = 'drive',
               omega_drive = omega_cont, bath_temp = T_cont)
a_in0 = Input('0', a, kappa_0, kind = 'bath', bath_temp = T_bath)

b_in0 = Input('0', b, gamma_m, kind = 'bath', bath_temp = T_bath)
```

And now we measure the spectrum and the linear response.

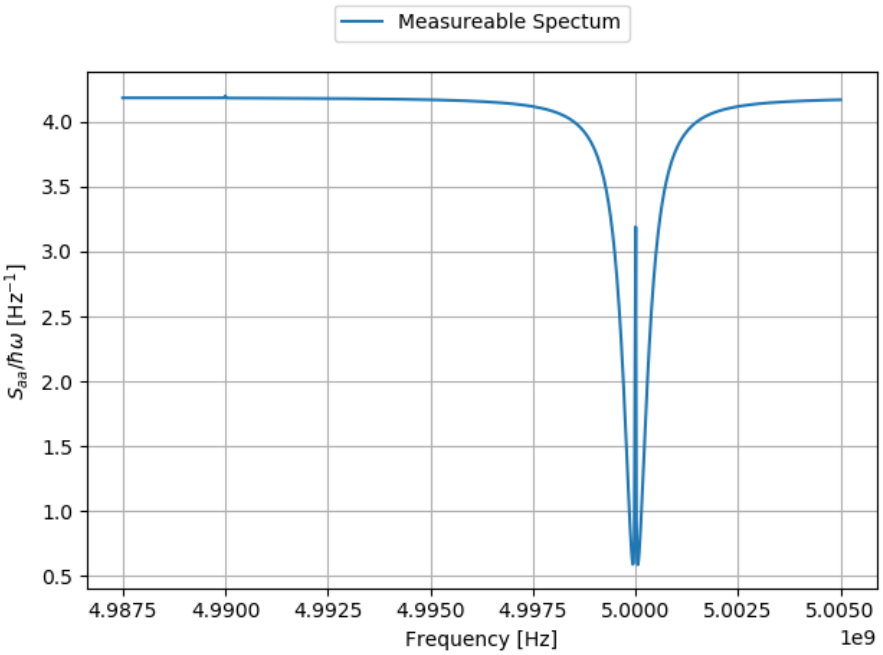

Figure 16: Optomechanical cavity output spectrum

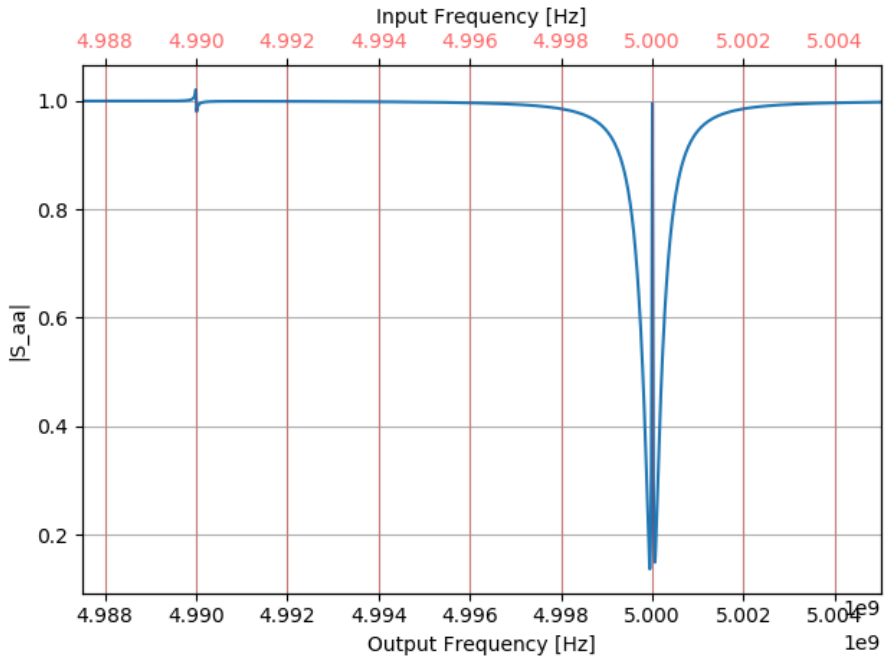

Figure 17: Optomechanical cavity linear response amplitude

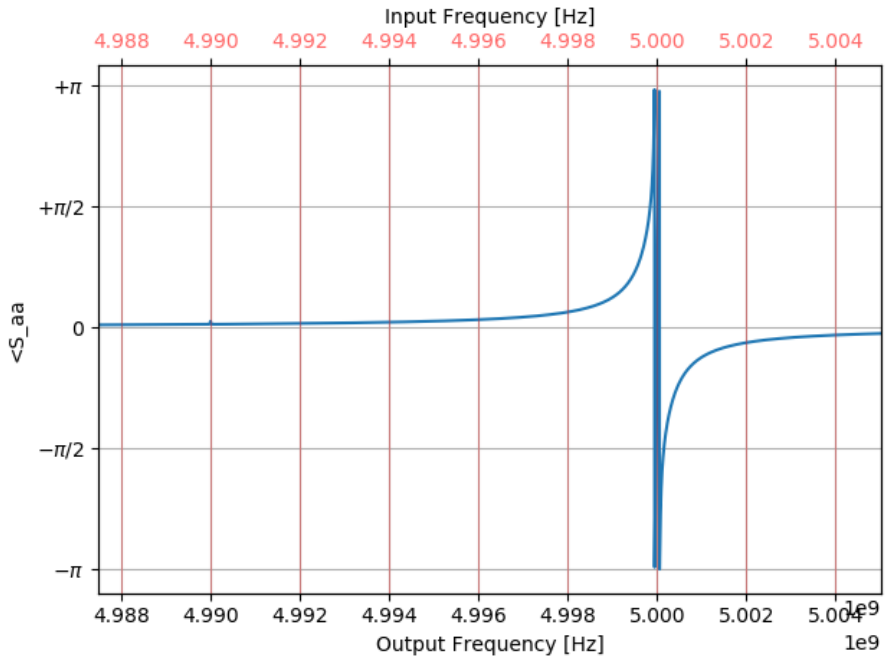

Figure 18: Optomechanical cavity linear response phase

These results can also be compared to theory. To see the details go to OMIT Test notebook. We can show for the transmission window

$$T = 1 - \kappa_{\text{ext}} \frac{\chi_{\text{opt}}(\Omega)}{1 + g^2 \chi_{\text{mech}}(\Omega)\chi_{\text{opt}}(\Omega)},$$

where

$$\chi_{\text{opt}}(\Omega) = [-i(\Omega + \Delta) + \kappa/2]^{-1}, \chi_{\text{mech}}(\Omega) = [-i(\Omega - \Omega_m) + \Gamma_m/2]^{-1},$$

with

$$\Delta = \omega_{\text{cont}} - \omega_{\text{cav}}, \quad \Omega = \omega_p - \omega_{\text{cont}}.$$

And the linear response from IOpy can be compared with this results as shown in figures bellow.

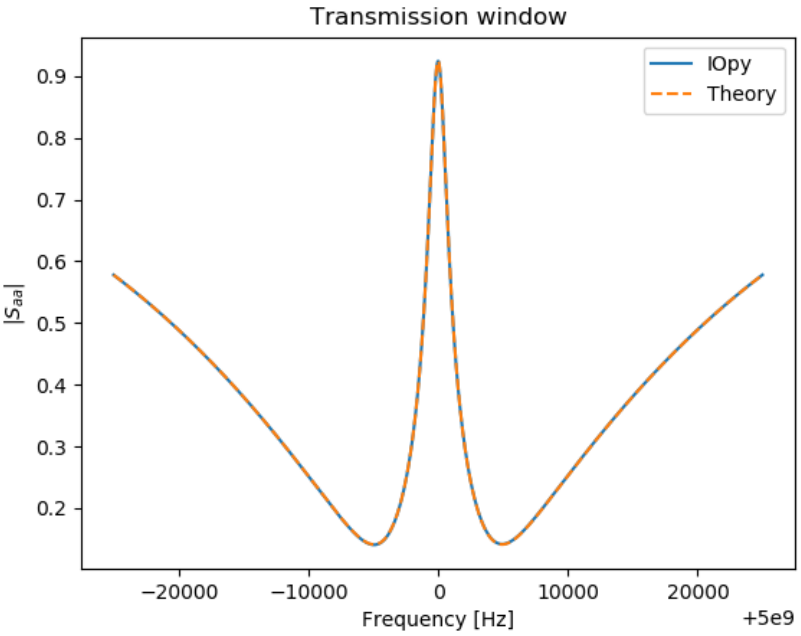

Figure 19: Optomechanical cavity linear response amplitude

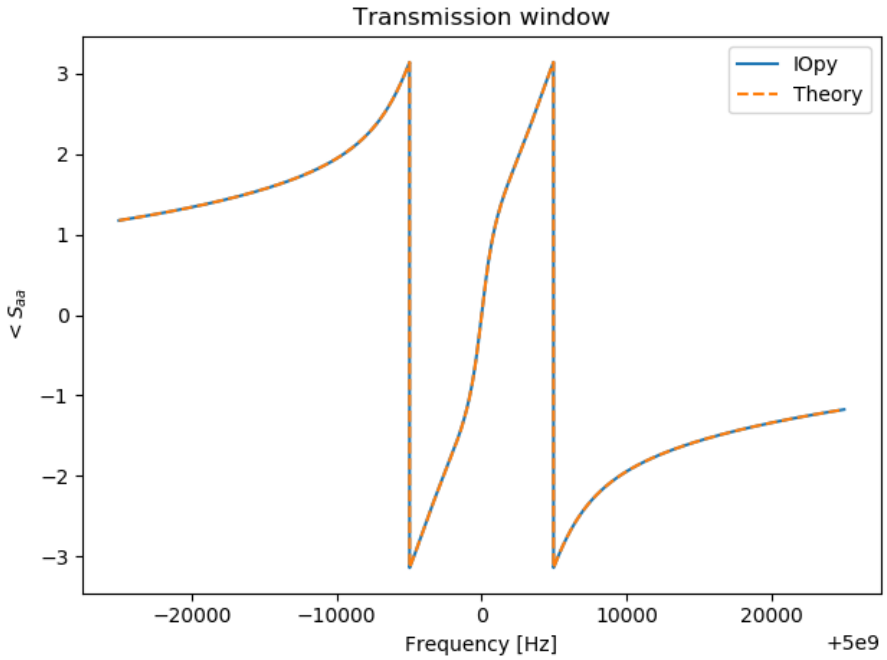

Figure 20: Optomechanical cavity linear response phase

Frequency Conversion Using OMIT

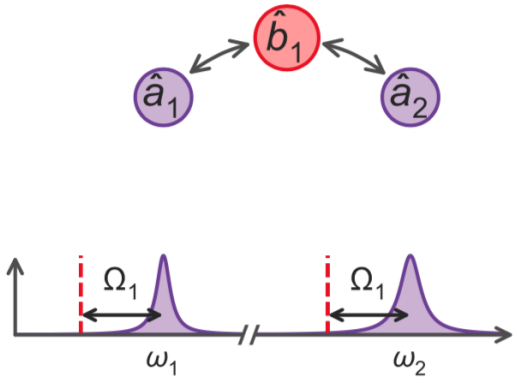

Figure 21: Reciprocal mechanically-mediated frequency conversion (adapted from Peterson et. al. 2018)

The last example is about the frequency conversion using the OMIT effect which is a step towards multimode calculations. In this scenario, two optical modes with different resonance frequencies are coupled to the same mechanical mode and the optical modes are each pumped on their red sidebands. If we inject a weak probe to one of the cavities, there will be an emission from the other cavity. The qualitative explanation is that when probe field is on resonance with the first cavity, it will beat with the pump and excite the mechanics. the excitation of the mechanics then phase modulates the second pump which creates sidebands. To see this effect in IOpy first we have to define the whole system as two optical modes each with one drive, tuned on their red sidebands and coupled to a shared mechanical mode.

```
omega_cav1 = 5e9*np.pi*2
kappa_01 = 0.3e6*np.pi*2
kappa_ex1 = 0.4e6*np.pi*2

omega_cav2 = 6e9*np.pi*2
kappa_02 = 0.3e6*np.pi*2
kappa_ex2 = 0.4e6*np.pi*2


omega_m = 5e6*np.pi*2
gamma_m = 100*np.pi*2

g_01 = 200*np.pi*2
g_02= 200*np.pi*2


P_in1 = 8e-10
Delta1 = -omega_m
omega_cont1 = omega_cav1 + Delta1
T_cont1 = 2
bath_temp1 = 10e-3

P_in2 = 8e-10
Delta2 = -omega_m
omega_cont2 = omega_cav2 + Delta2
T_cont2 = 2
bath_temp2 = 10e-3

from DCnonlinearities import optomechanics

omdir1 = optomechanics(P_in1, kappa_01, kappa_ex1,
                       omega_cav1, omega_cont1, omega_m, g_01)
g1= omdir1['g']
omega_cav1 = omdir1['omega_c']
```

```
omdir2 = optomechanics(P_in2, kappa_02, kappa_ex2,
                       omega_cav2, omega_cont2, omega_m, g_02)
g2= omdir2['g']
omega_cav2 = omdir2['omega_c']



a1 = Mode('a1', omega_cav1)
a2 = Mode('a2', omega_cav2)
b = Mode('b', omega_m)

a_cont1 = Input('ex', a1, kappa_ex1, kind = 'drive',
                omega_drive = omega_cont1, bath_temp = T_cont1)
a_in01 = Input('0', a1, kappa_01, kind = 'bath', bath_temp = bath_temp1)

a_cont2 = Input('ex', a2, kappa_ex2, kind = 'drive',
                omega_drive = omega_cont2, bath_temp = T_cont2)
a_in02 = Input('0', a2, kappa_02, kind = 'bath', bath_temp = bath_temp2)

b_in0 = Input('0', b, gamma_m, kind = 'bath', bath_temp=10e-3)

g_a1b = Coupling(a1, b, g1 * np.array([1,0,1,0]))
g_a2b = Coupling(a2, b, g2 * np.array([1,0,0,0]))

sys_om = System([a1, a2, b], [a_in01, a_in02, b_in0, a_cont1, a_cont2],
                [g_a1b, g_a2b])
```

Then we have to define the output ports and then measure the linear response from the input drives port of first cavity to the output port of the second cavity.

```
a_outex1 = Output(sys_om, a_cont1)
a_outex2 = Output(sys_om, a_cont2)

omegas = np.linspace(omega_cav1 - 2.5* omega_m, omega_cav1 +  2.5 * omega_m, 10000)
omegas_new, A = me.linear_response(omegas, sys_om, a_outex2, a_cont1, plot = True)
```

And the result, which is a measure of the frequency conversion efficiency, can be seen in the graph below.

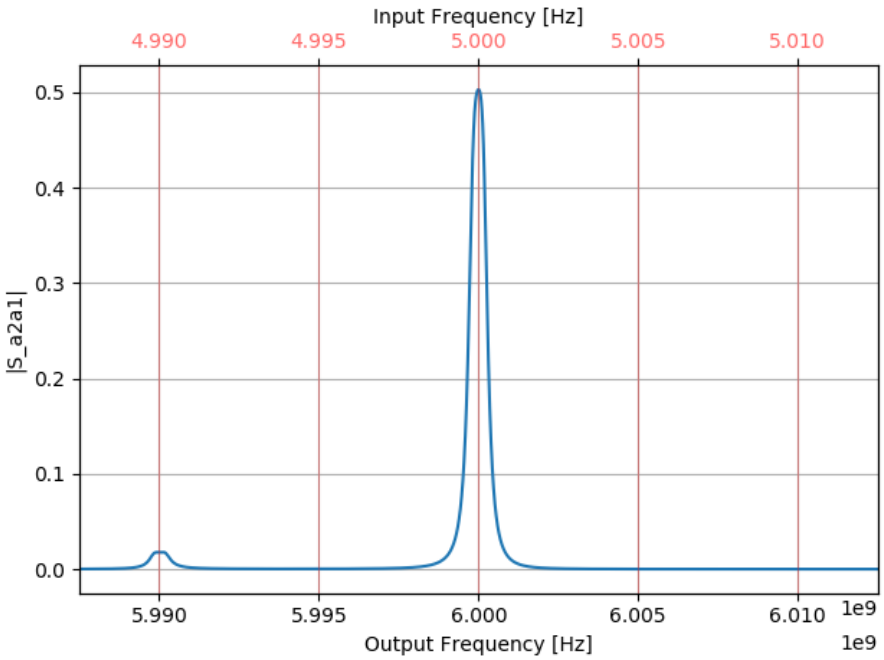

Figure 22: Frequency conversion linear response amplitude

Both the input field and output field frequencies are shown on the graph. This means every point on this curve is the amplitude of the output field at the "output frequency", when the input probe is at the corresponding "input frequency".

Outlook

Multimode optomechanics

One of the biggest motivations for starting the IOpy project was to develope a computational tool to help the experts in the field with the theoretical calculations. When the optomechanical systems become more complex the calculations also get more complicated and even impossible to do by hand without approximations. Especially, when the system invloves more oscillating modes like optical cavities, mechanical oscillators or driving fields. Multimode optomechanics, a growing field of interest, can be improved if such computational tools has the power to do the multimode calculations with an acceptable accuracy. But the barrier for the IOpy to do these calculations is arising from almost the same issue as when we want to drive an optical mode with more than one pump.

Multidrive Issue

In many experiments in optomechanics we need to drive optical resonators with more than one pump field (for example see Suh et. al. 2014). In this case the theoretical calculations are not as clear as the simple case where each optical mode is driven by a single pump. The complexity is arising from the fact that due to the nonlinear nature of the optomechanical interaction, there would always be frequency mixing terms in the dynamics. In case of a single pump these mixings can be divided into fast and slowly varying terms and by going to the rotating frame of the drive, the equations of motions for the slowly varying terms become time independent and with linearizing the equations in principle they can be solved exactly in the frequency domain. But in case of multiple drives, because of the presence of multiple powerful tones, in general it is impossible to apply the same idea and equations of motion can not become time independent. In another language, there no longer exists a single rotating frame in which every other time variation can be considered slowly varying. The idea of having different rotating frames also fails because of the nonlinearities that prevents two rotating frames to be independent of each other.

However, there are some approaches to resolve the problem still in frequency domain. For example, in the case of driving a mode with two pumps one solution is to use the Floquet ansatz to expand the variables by a Fourier series with principle frequency equal to difference frequency of the two drives (see Malz and Nunnenkamp, 2016). As another example, in the context of a four-mode isolator, where again each optical cavity is pumped with two drives to couple to two different mechanical modes simultaneously, the idea is to define the higher order mixing terms as "auxiliary modes" and expand the model to a model with higher number of modes as shown in the picture below from the work of Peterson et. al., 2018.

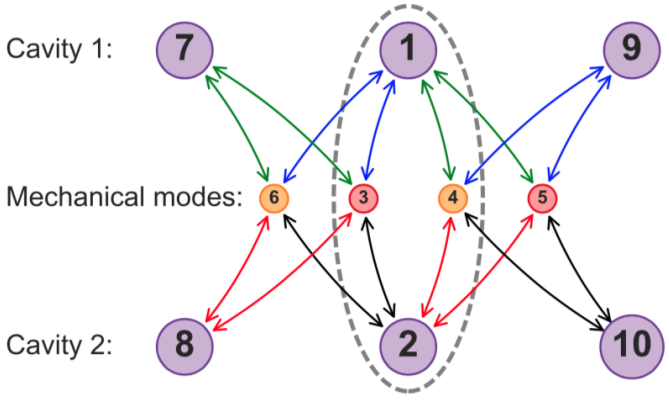

Figure 23: The idea of the expanded mode basis for the four-mode isolator (Peterson et. al., 2018).

But in both cases, the calculations rely on truncating an infinite series and using the approximations like rotating wave approximation which we want to avoid in IOpy for keeping the calculations general. Moreover, these ideas will result into huge complexity in case of more than two drives.

Time Domain Simulations

These issues lead us to think about a totally different approach for the calculations which is adopted from what is really happening in the experiments. In experiments, a VNA measures the scattering matrix of a network injecting a coherent signal (a sine wave) to the network and extracting the amplitude and phase from the output of the network. This, we can exactly done with time-domain simulations. In principle, we can write the most general Hamiltonian of the optomechanical network as the following

$$H_{\text{sys}} = \sum_k \hbar\omega_{\text{cav},k} a_k^\dagger a_k + \sum_j \hbar\Omega_j b_j^\dagger b_j - \hbar \sum_{j,k,l} [g_0]_{kl}^j a_k^\dagger a_l (b_j^\dagger + b_j),$$

and

$$H_{\text{drive}} = \sum_{j,m} i\hbar\sqrt{\kappa_{jm}}(s_{\text{in},jm}(t) a_j^\dagger e^{-i\omega_m t} + H.c.).$$

With this, we can derive the equations of motion in the time domain and we can define the output ports with the input-output theory as $s_{\text{out},jm} = s_{\text{in},jm} - \sqrt{\kappa_{jm}} a_j$. Then for calculating the scattering matrix we can simulate the equations of motion in the time domain for inputs as sine waves, spanning a frequency band. Then look at the Fourier transform of the output wave at the frequencies we are interested to extract the amplitude and phase of the response. Please note this scattering matrix will not be a simple scattering matrix anymore. Because for example for a single weak sine wave there can be multiple sine waves at the output but we can still look for the frequencies of interest. Once the scattering matrix is calculated, every other parameter like the spectrum can be calculated.

This simulation is done for a simple case of a single cavity. The results for this simulation are shown in the figures below.

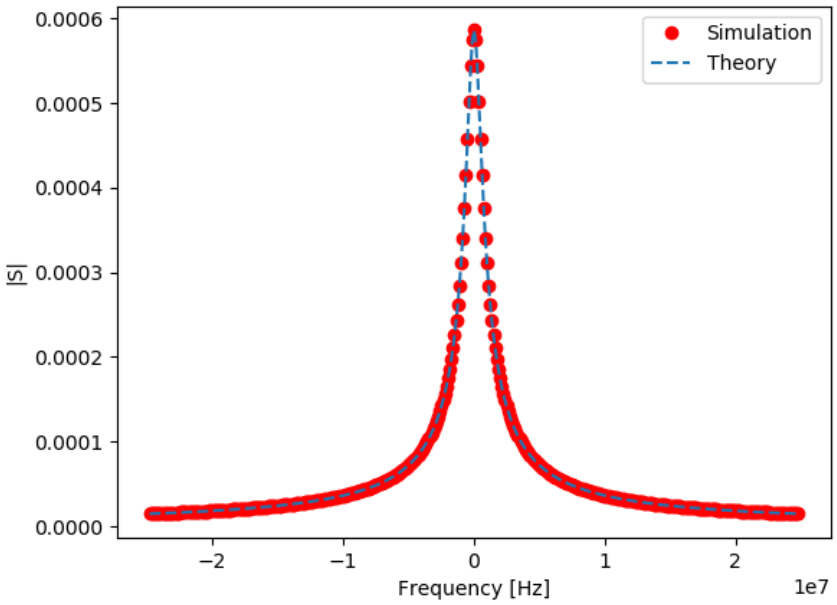

Figure 24: The amplitude of the linear response calculated with time domain simulation

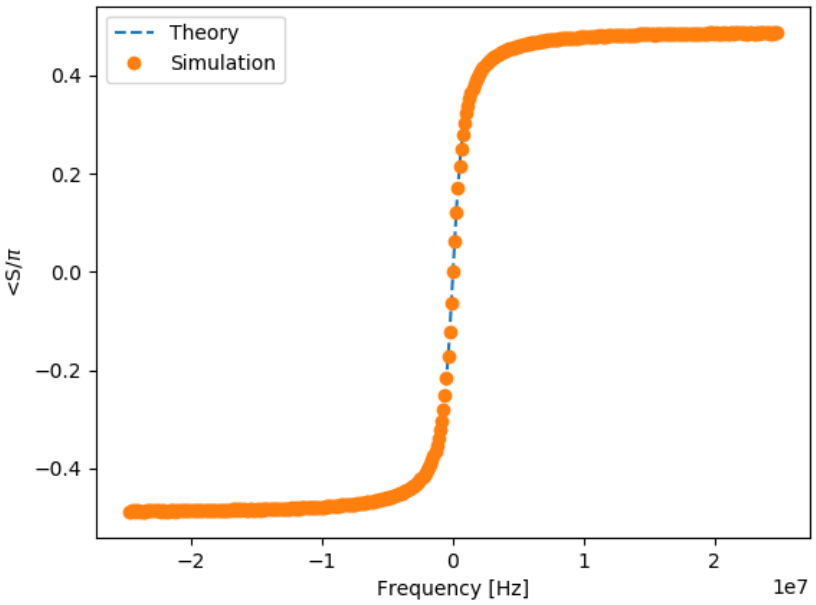

Figure 25: The amplitude of the linear response calculated with time domain simulation

Although this idea works for the simple cavity, we have to further develop the code in order to cope with more complex systems like the basic optomechanics. Developing a time domian calculation library for the IOpy is one of the main goals of the future.

Beyond Optomechanics

Another future goal for IOpy is to use it for physical systems other than optomechanics. In IOpy calculations, the physical nature of the problem is not really taken into account. For example all the examples given so far are in microwave domain (for superconducting circuit optomechanics) but all of them can also be implemented for the optical domain. In principle, we can go even further. As it can be inferred from the package name (IO in IOpy stands for Input-Output) our goal is to develope IOpy in a way that it is able to simulate most phenomenon which deal with coupled oscillators that can be formulated in input-output formalism.

References

Agarwal, G.S., and S. Huang, 2010, Phys. Rev. A 81, 041803(R).

Aspelmeyer, M., T.J. Kippenberg, and F. Marquardt, 2014, Rev. Mod. Phys. 86, 1391.

Clerk, A. A., F. Marquardt, and J. G. E. Harris, 2010, Phys. Rev. Lett. 104, 213603.

Fleischhauer, M., A. Imamoglu, and J.P. Marangos, 2005, Rev. Mod. Phys. 77, 633

Gardiner, C. W., and P. Zoller, 2004, Quantum Noise, Springer Series in Synergetics (Springer, Berlin/Heidelberg).

Malz, D., and A. Nunnenkamp, 2016, Phys. Rev. A 94, 023803.

Peterson, G.A., F. Lecocq, K. Cicak, R.W. Simmonds, J. Aumentado, and J.D. Teufel, 2018, Phys. Rev. X 7, 031001.

Schliesser, A., 2009, "Cavity optomechanics and optical frequency comb generation with silica whispering-gallery-mode resonators," Ph.D.thesis(Ludwig-Maximilians-UniversitatMunchen).

Suh, J., A. J. Weinstein, C. U. Lei, E. E. Wollman, S. K. Steinke, P. Meystre, A. A. Clerk, and K. C. Schwab, Science 344, 1262 (2014).

Teufel, J. D., D. Li, M. S. Allman, K. Cicak, A. J. Sirois, J.D. Whittaker, and R.W. Simmonds, 2011, Nature (London) 471, 204.

Weis, S., R. Riviere, S. Deleglise, E. Gavartin, O. Arcizet, A. Schliesser, and T. J. Kippenberg, 2010, Science 330, 1520.