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) apear 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 soultions of Langevin equations in different optomechanical system can result into really novel phenomena. However,

the procedure to find these solutions is almost the same for all of them (input-output formalism, which the name IOpy is also coming from it) and the essential difference between them is the difference between the physical setups. Moreover, in complex setups the calculations for finding the solutions can be really hard and tedious to do by hand (for example inverting matriced with large dimensions).

On the other hand, in many problems in optomechanics there are a lot of elements which are involved in the dynamics. For people who are not porfessional in the field, like students who want to study optomechanics, as a young field of research it can be confusing for them to distinguish different effects of different elements involved in the dynamics. Looking for a solution to resolve the two mentioned issues was the motivation for writing 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 visualise the results in way can help people to test their theoretical results and also help begginers to grasp the elements of the optomechanics. For example to see the spectrum of 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 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, 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:

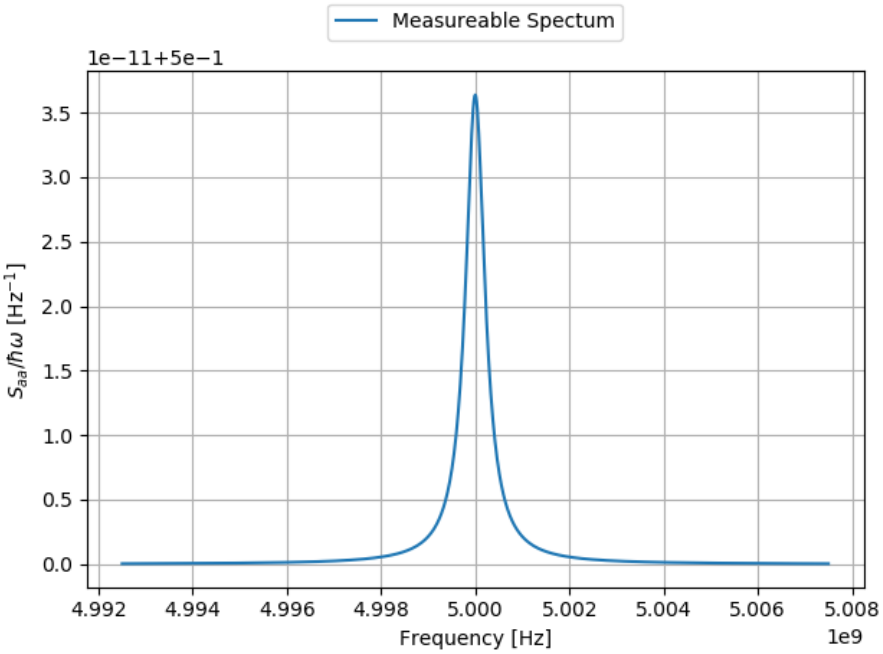

Figure 1: Simple cavity output spectrum

A more detailed explenation of this example as well as more examples for optomechanics are availabe on the Examples page.

Structure of IOpy

IOpy is made of four scripts which each of them has a special purpose:

`elements`: For defining different components of the physical system (modes, couplings, input-output field etc.)

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

`measurement`: For calculations on the output fields like linear response and spectrum.

`plots`: For visualising the measurements results on graphs.

Installation

Theory

Input-output formalism

In this section, the theory behind the IOpy calculations is introduced. There are essentially two main approaches through modeling of open quantum systems. First is density matrix approach and using master equations and second is using the Langevin equations, which is also known as input-output theory. Input-output theory allows us to directly model the quatum fluctuations injected from any couplings port into the system. Quantum Langeving equations are formulated on the level of 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 enviroment through the dissipation rate of κ_0 the equations of motion can be written as:

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

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

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

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:

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

Or in terms of quadratures:

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

In general, any set of Langevin equations can in principle be linearized around the stationary solutions and can be written in a simillar way:

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

With the input-output relations:

$$Z_{out} = Z_{in} - \mathbf{L}^T Z \tag{2}$$

Where Z is a vertical vector containing the quadreture operators of different oscillatros involved in the dynamics and Z_{in} and Z_{out} are vectors containing the qudreture operatos of the input and output fields. For the optical cavity of the example above we have:

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

And:

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

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:

$$Z_{out}(\omega) = \mathbf{S}(\omega) Z_{in}(\omega) \tag{3}$$

Where:

$$\mathbf{S} = \nVdash + \mathbf{L}^T (i\omega + \mathbf{M})^{-1} \mathbf{L} \tag{4}$$

Measurements

In IOpy calculations there are two ways to measure the output fields, which are essentially same as what we do in practice. First is the linear response of the system, which is simillar to output of a VNA. And second is the spectrum of the output field, which is the same as the ouput of the spectrum analyser.

Linear response

According to the Equation (3), 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:

$$\begin{pmatrix} X_{out,i} \\ Y_{out,i} \end{pmatrix} = \mathbf{S}^{(ij)} \begin{pmatrix} X_{in,j} \\ Y_{in,j} \end{pmatrix} \tag{5}$$

Then the output field operator can be written as:

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

In many cases the input field is a coherent drive with constant amplitude. Therefore, $X_{in,j}$ and In many cases the input field is a coherent drive with constant amplitude. Therefore, $X_{in,j}$ and

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

$$a_{out,i} = (\mathbf{S}_{11}^{(ij)} + i\mathbf{S}_{21}^{(ij)})a_{in,j} \tag{6}$$

or in a more familiar notation:

$$\chi^{(ij)}(\omega) = \mathbf{S}_{11}^{(ij)} + i\mathbf{S}_{21}^{(ij)} \tag{7}$$

Spectra

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

$$Q(\tau) = \langle Q_{ij} Z_{out,i}(0) Z_{out,j}(\tau) \rangle \tag{8}$$

The Fourier transform of this correlator would be the spectral density or spectrum.

$$S_{QQ}[\omega] = \int_{-\infty}^{\infty} Q(\tau) e^{i\omega\tau} d\tau \tag{9}$$

It can be easily shown with a simple calculation the we can rewrite the spectrum in the following way:

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

According to Equation (3), we can write the output signals in terms of input signals:

$$Z_{out,i}(\omega) = S_{ik}(\omega) Z_{in,k}(\omega)$$
$$S_{QQ}[\omega] = \frac{1}{2\pi} Q_{ij} \int_{-\infty}^{\infty} S_{ik}(\omega_1) S_{jl}(\omega) \langle Z_{in,k}(\omega_1) Z_{in,l}(\omega) \rangle d\omega_1 \tag{10}$$

In general, the input signals can be correlated to each other and have complecated statistical behaviours, but based on what realy happens in experiments it's a good approximation to (1) consider different sources to be uncorrelated and (2) consider each input source a white noise source. More precisely:

$$\langle Z_{in,k}(\omega_1) Z_{in,l}(\omega) \rangle = 2\pi \delta_{kl} \mathcal{S}_k^{in}(\omega) \delta(\omega_1 + \omega) \tag{11}$$

Where $\mathcal{S}_k^{in}(\omega)$ is the spectral density of the kth input signal. Now we can simplify Equation (10) :

$$S_{QQ}[\omega] = Q_{ij} S_{ik}(-\omega) S_{jk}(\omega) \mathcal{S}_k^{in}(\omega) \tag{12}$$

This is the final result that we use in our calculations. Here we make a remark on the summations by rearanging the terms in the following way:

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

In this way, the measured spectrum can be seen as summation of different contributions from different noise sources.

Library

In this section different classes, functions and other commands of IOpy are introduced. IOpy has 4 main scripts including `elements`, `measurement`, `DCnonlinearaity` and `plots`.

elements

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

Class `Mode`

Harmonic oscillators of any kind can be defined using this class.

```
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`

Input field coupled to a `Mode`. Inputs can be coherent drives (pumps) or thermal baths.

```
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: tempreture 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 approximatley take
    the thermal spectrum to be flat 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.
```

Class `Coupling`

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:

$$H_{int} = 4\hbar(V_{g,1}q_1q_2 + V_{g,1}q_1p_2 + V_{g,1}p_1q_2 + V_{g,1}p_1p_2)$$

For example 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 other wise.
```

Class `System`

Defines the complex system made of coupled `Mode`s and `Input`s. It's most important purpose of it is for calculating 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) for more about 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):
    Constructing 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 (4) for more about scattering matrix.

Class `Output`

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 for measuring the output fields. Different measurement schemas can be used and linear response or spectrum of the output fields can be seen as results.

Class `MeasurementOperator`

(it seems this one is useless!)

Class `PowerMeasurement`

A power measurement scheme object. The correlator function and measurement matrix in this scheme are:

$$Q(\tau) = \langle q(0)q(\tau) + iq(0)p(\tau) - ip(0)q(\tau) + p(0)p(\tau)\rangle$$

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

```
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 measurment matrix.
```

Class `HomodynMeasurement`

A Homodyn measurement scheme object with a theta phase. The correlator function and measurement matrix in this scheme are:

$$Q(\tau) = \langle \cos^2(\theta)q(0)q(\tau) + \sin(\theta)\cos(\theta)q(0)p(\tau) + \sin(\theta)\cos(\theta)p(0)q(\tau) + \sin^2(\theta)p(0)p(\tau)\rangle$$

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

```
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 measurment matrix.
```

Function `linear_response`

The linear response (susceptibility) of the system from one specific input port to an output port in frequency domain:

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

Refer to Equation (7) for more about linear response.

```
Args:
    Omegas: the frequencies vector we want to claculate
            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 claculate
                the linear response for them, in frame of the
                output field (not a rotating frame)
    a: the susceptibility we want to measure.
```

Function `spectrum`

The spectrum of an output field. Refer to spectra section for more about the spectra.

```
Args:
    omegas: the frequencies vector we want to claculate the
            spectrum for them, in frame of the output field
            (not a rotating frame)
    measurement: the measurement scheme, of kinds PowerMeasurement
                 or HomodynMeasurement.
    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.
```

DCnonlinearities

Functions in this script are used for calculating the DC shifts resulting form nonlinear effects.

Function `Kerr_effect_nbar`

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

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

```
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 route of the third order polynomial equation.
```

Function `optomechanics`

This function findes the steady state average number of photons 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:

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

```
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 are for plotting the linear responses and spectrums.

Function `plot_linear_response`

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

```
    Args:
        omegas: the vector containing the frequecnies
                in rad/sec (not in a rotating frame).
        A: 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 for plotting the spectra.

```
    Args:
        omegas: the vector containing the frequecnies
                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 famous phenomena in optomechanics context are presented. These calculations are first as examples of using IOpy in simulating Langevin equations, and second are benchmarks which results of IOpy can be compared against results which are theoretically developed.

Simple Cavity

The first example is simulating a hot microwave resonator. Here the temperature of the bath is higher than the temperature of the drive and therefore we would expect to see an emmition shaped like a Lorenzian. To see the fool written example, go to Simple cavity example. first we have to define 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 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 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)
```

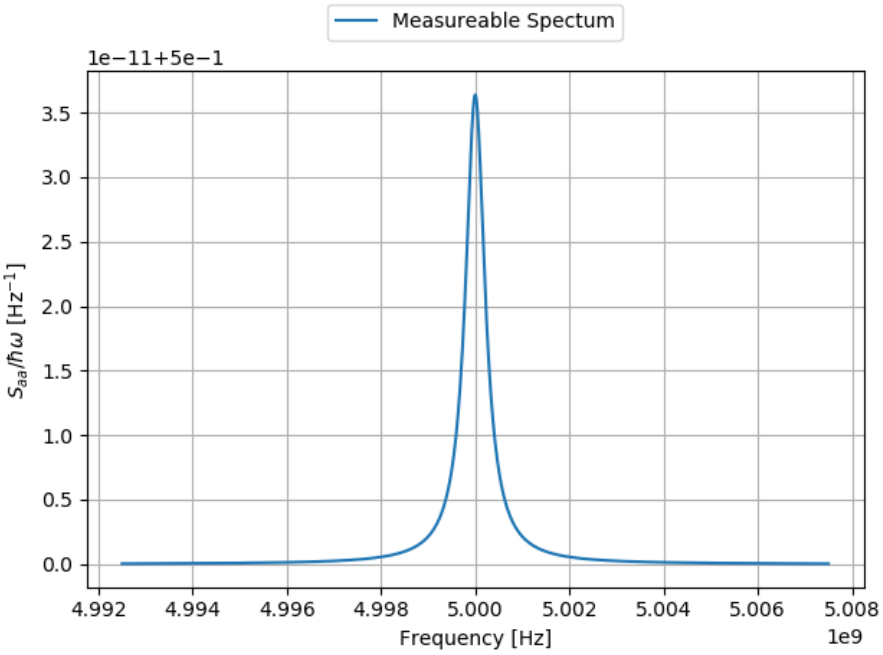

Figure 2: Simple cavity output spectrum

As we expect, we can see a Lorentzian in the spectrum. If the tempreture 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)
```

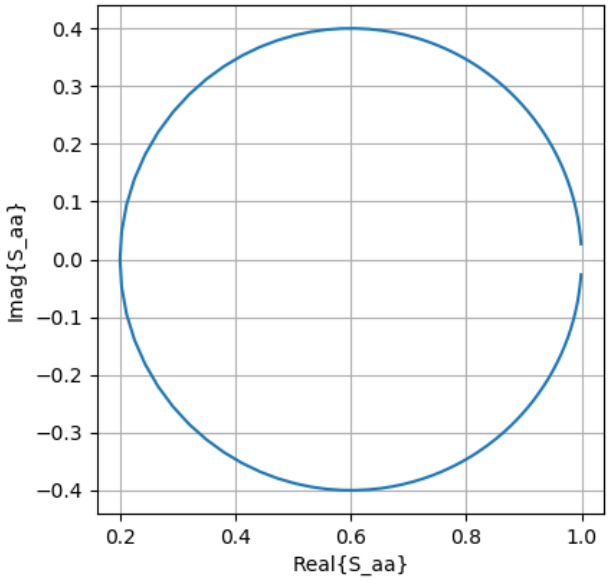

Figure 3: cavity linear response in complex space

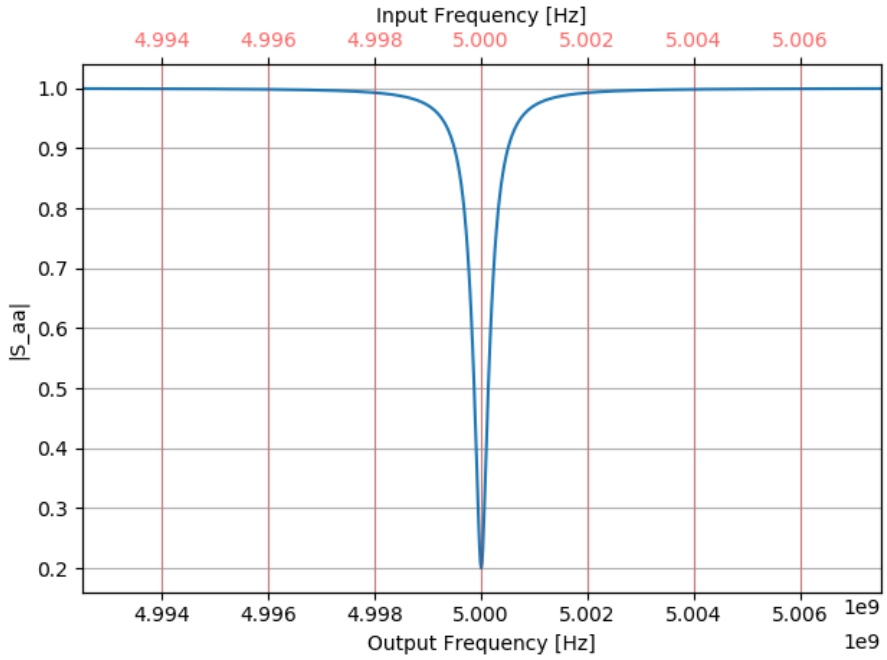

Figure 4: cavity linear response amplitude

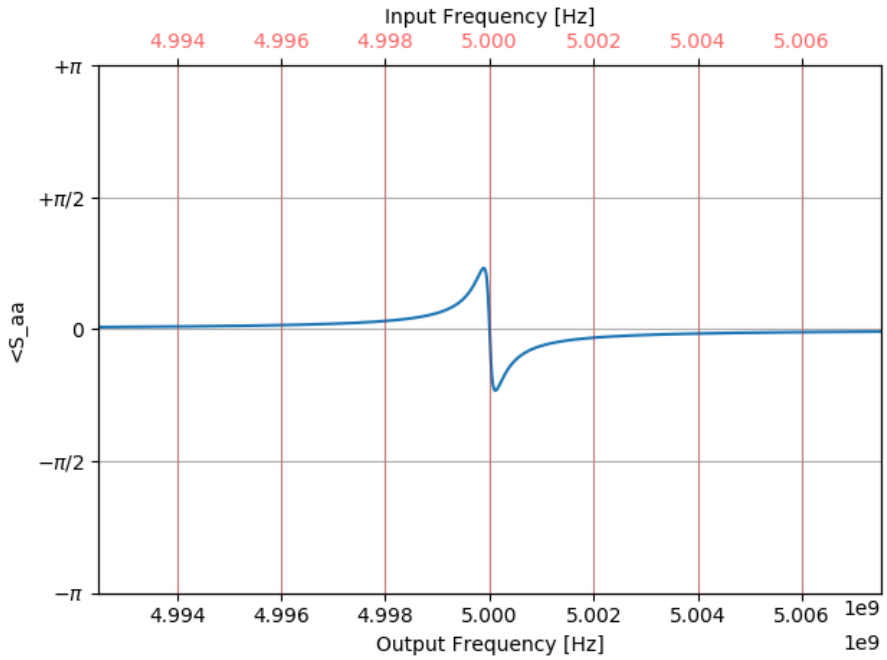

Figure 5: cavity linear response phase

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.

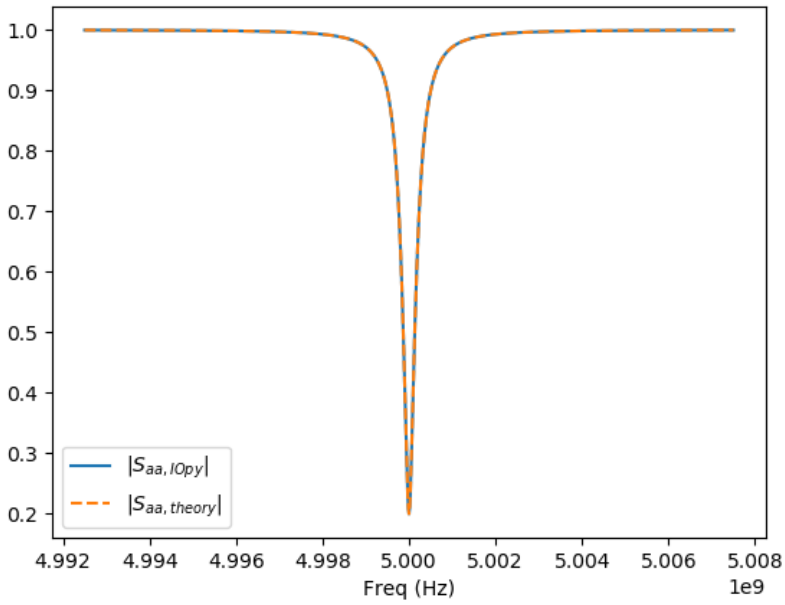

Figure 6: cavity linear response amplitude

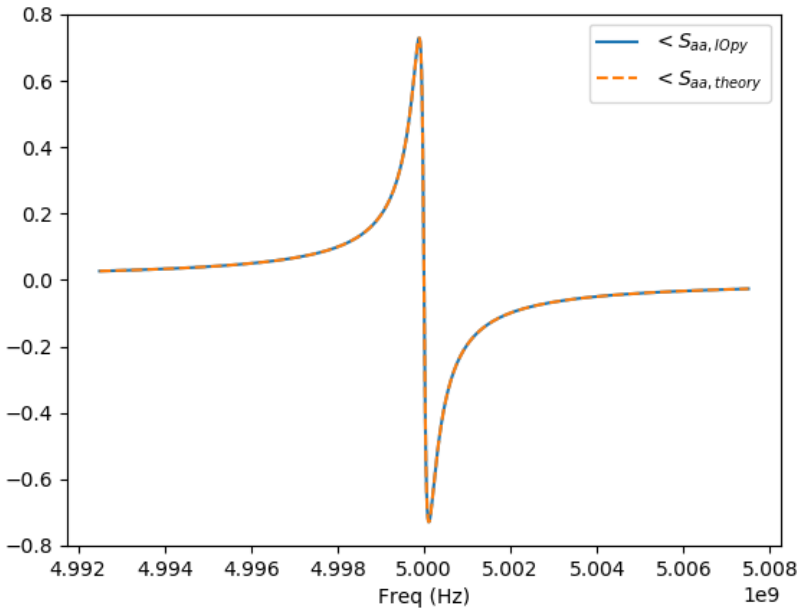

Figure 7: cavity linear response phase

Basic optomechanics and cooling

In this example we simulate an optomechanical system with a weak drive. Here we want to see the optomechanical cooling due to increase in optomechanical damping rate. To see the fool written example, go to Basic optomechanics.

In the basic optomechanical interaction, the cavity resonance frequency shift by a constant value due to DC nonlinearity. Therefore, before difining 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 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.

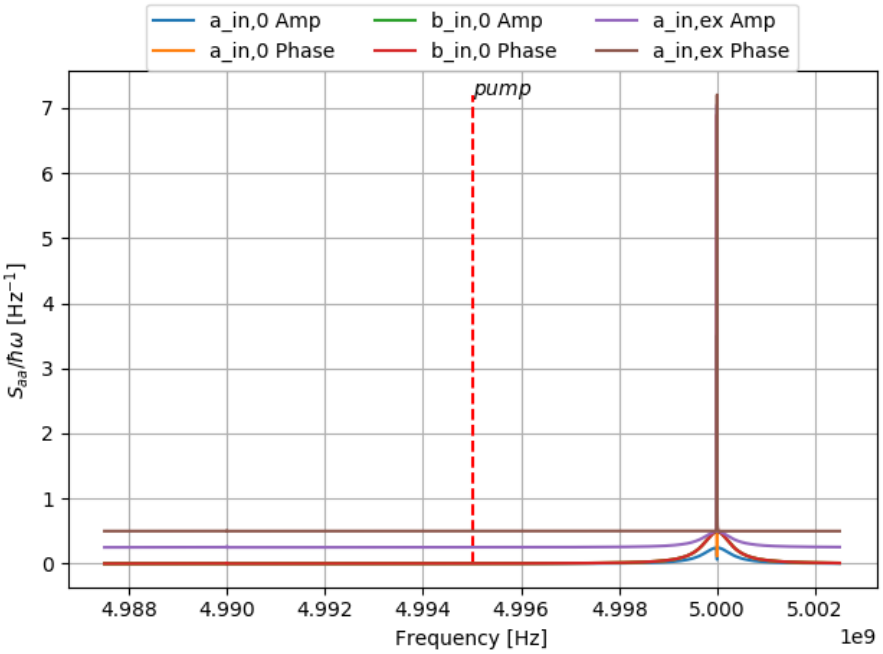

Figure 8: optomechanical cavity output spectrum

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

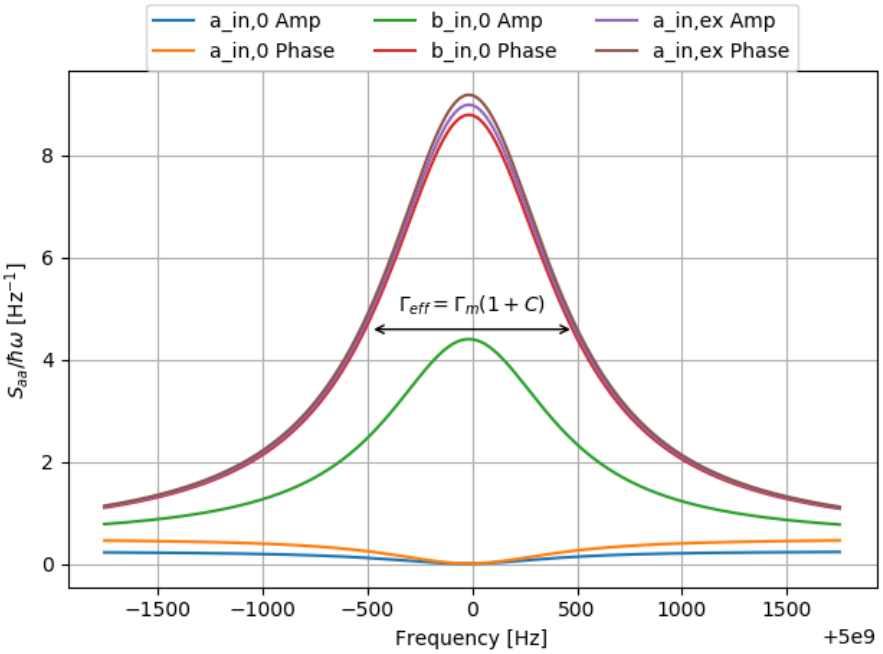

Figure 9: optomechanical cavity output spectrum

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

To this end, the code is exactly the same as the previous example but with a different input power. To see the full code go to Strong coupling regime.

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

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

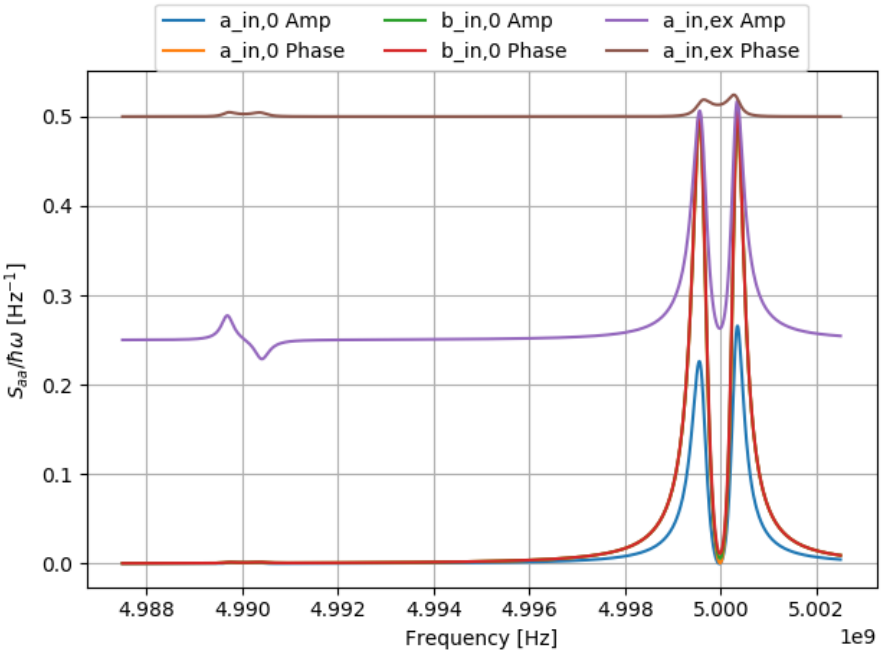

Figure 10: optomechanical cavity output spectrum

To see better the splitting we change the measurement frequencies:

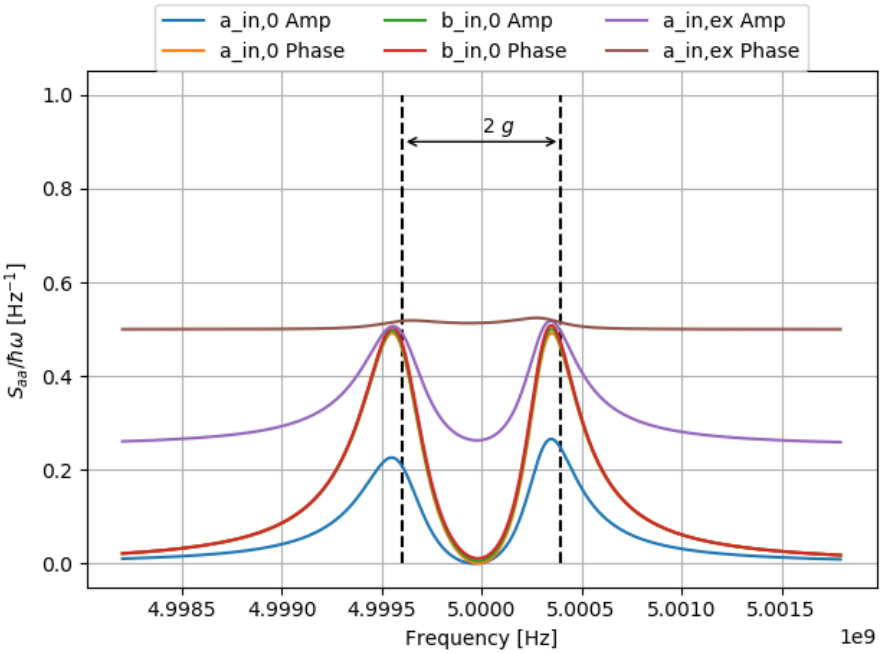

Figure 11: optomechanical cavity output spectrum

Optomechanically induced transparency

This is example is about the optomechanically induced transparency also known as OMIT. This effect was observed in atoms (electromagnetically induced transparency Fleischhauer, Imamoglu, and Marangos, 2005) as the cancellation of absorbtion in the presence of an auxiliary laser field. OMIT was predicted theoritcally by Schliesser, 2009 and Agarwal and Huang 2010. When the optical cavity is pumped on the red sideband, if 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 simillar to the Basic optomechanics example. The difference here is that we set the driving field (control field, in this context) to be a high tempreture source. In this way, the noise around this control field play the roll of the weak probe field for us, so we can see the OMIT effect in the spectrum, as well as the linear response. To see the full code go to OMIT example.

```
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.

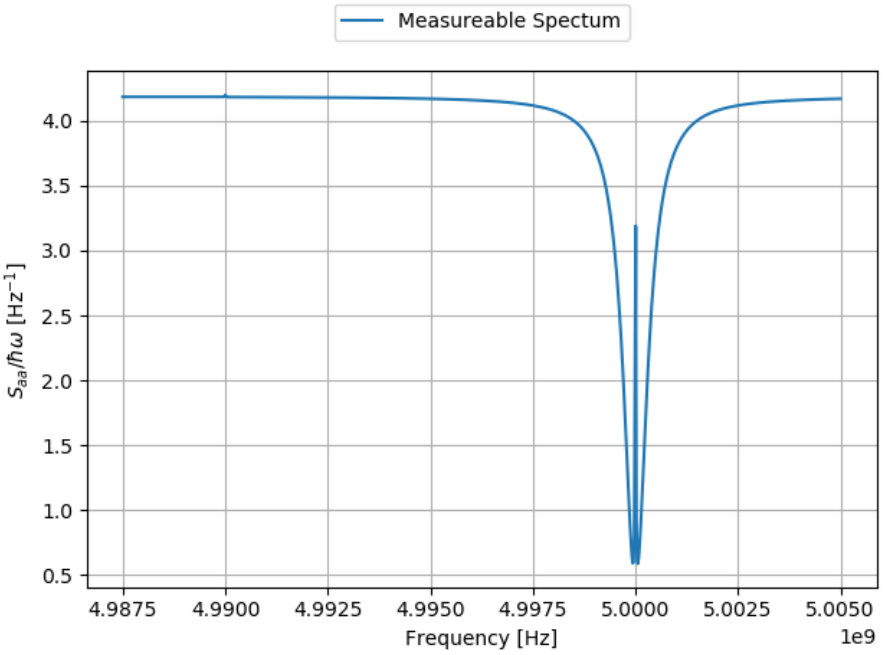

Figure 12: optomechanical cavity output spectrum

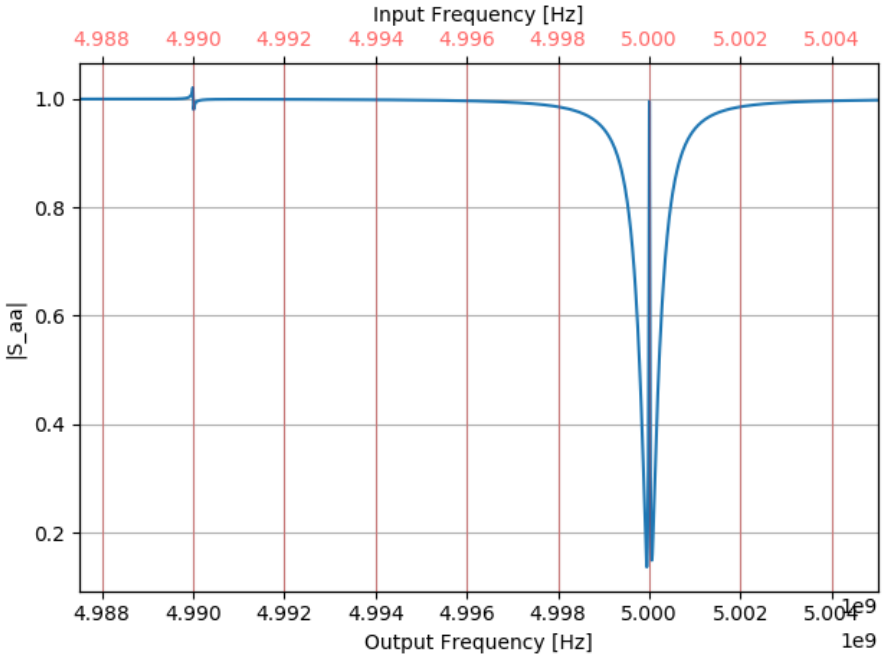

Figure 13: cavity linear response amplitude

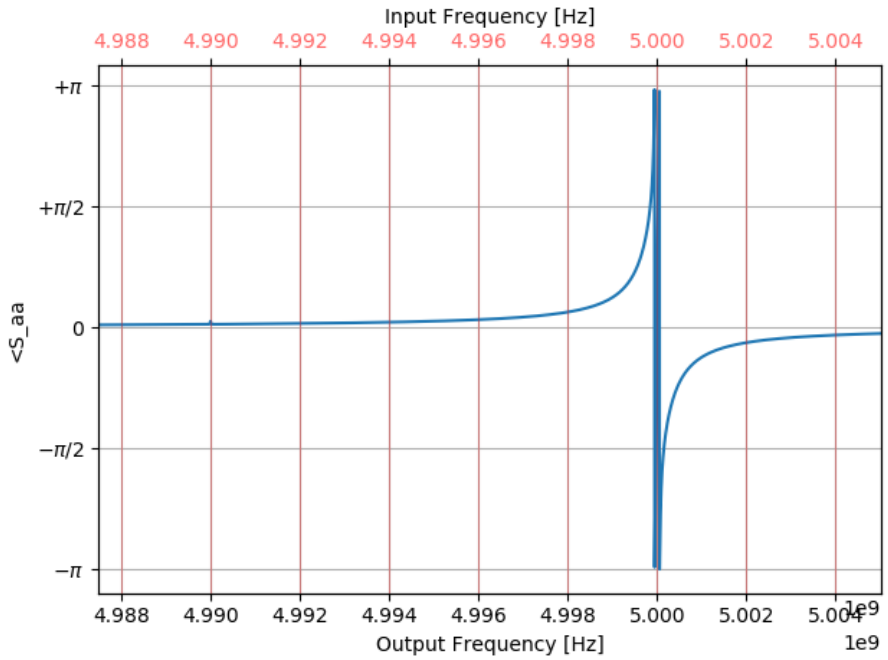

Figure 14: cavity linear response phase

These results can also be compared to theory. To see the details go to OMIT Test notebook. By a calculation we can show for the transmission window:

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

Where:

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

With:

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

And the linear response from IOpy can be compared with this results:

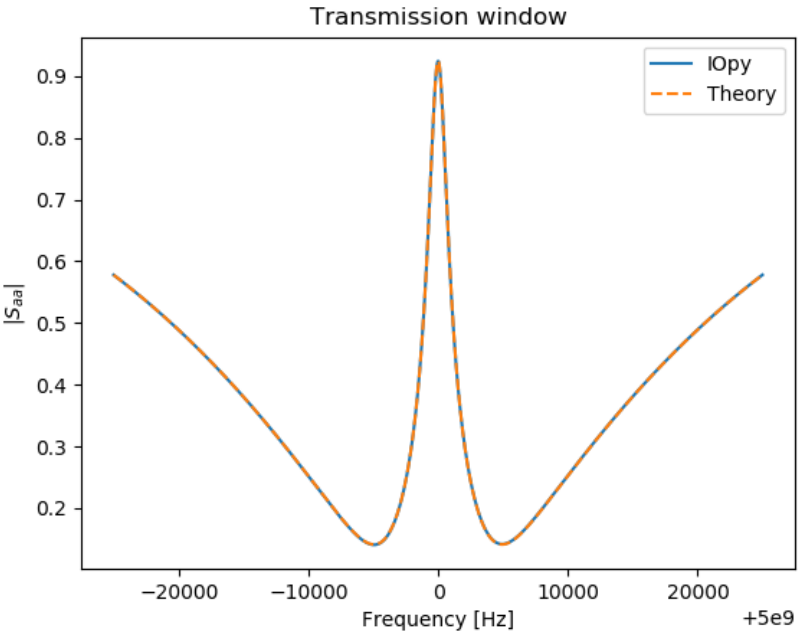

Figure 15: cavity linear response amplitude

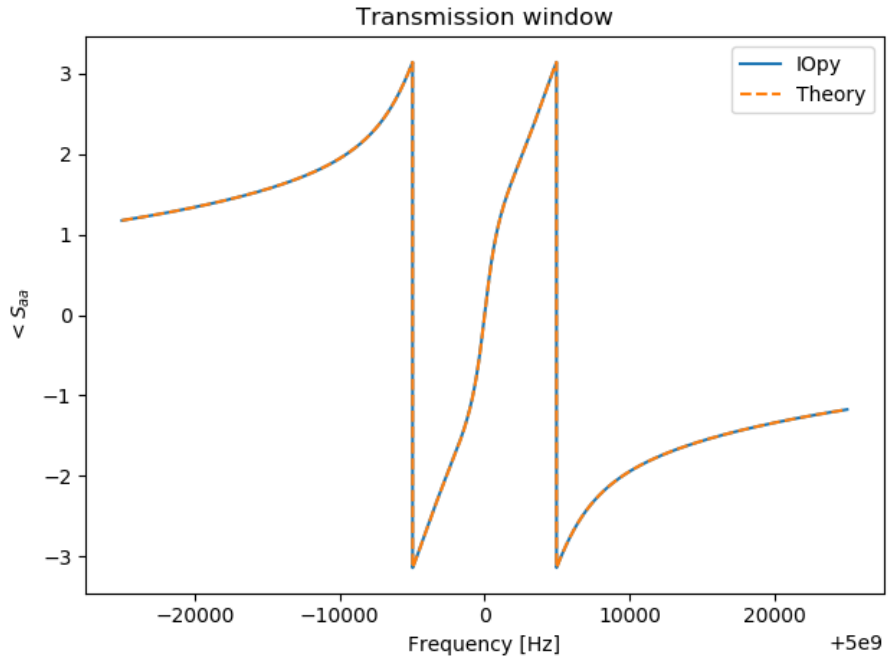

Figure 16: cavity linear response phase

Frequency conversion using OMIT

The last example is about the Frequency Conversion using OMIT effect, which is a step through multimode calculations. In this scenario, two optical modes with different resonance frequencies are two the same mechanical mode and the optical modes are pumped on their red sidebands. If we inject a weak probe to one of the cavities, when this probe is around resonance with this cavity there will be an emission from the other cavity, around its resonance frequency. 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. Then on the other cavitu beatting of the mechanics and the pump will excite a field which it's frequency is spaced from the second pump by the value of the mechanical resonance frequencies. These excitations are both at red and blue sides of the pump, which the blue side would be the resonance frequency of the second cavity.

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. To see the full code, go to Frequency Conversion using OMIT.

```
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 would be:

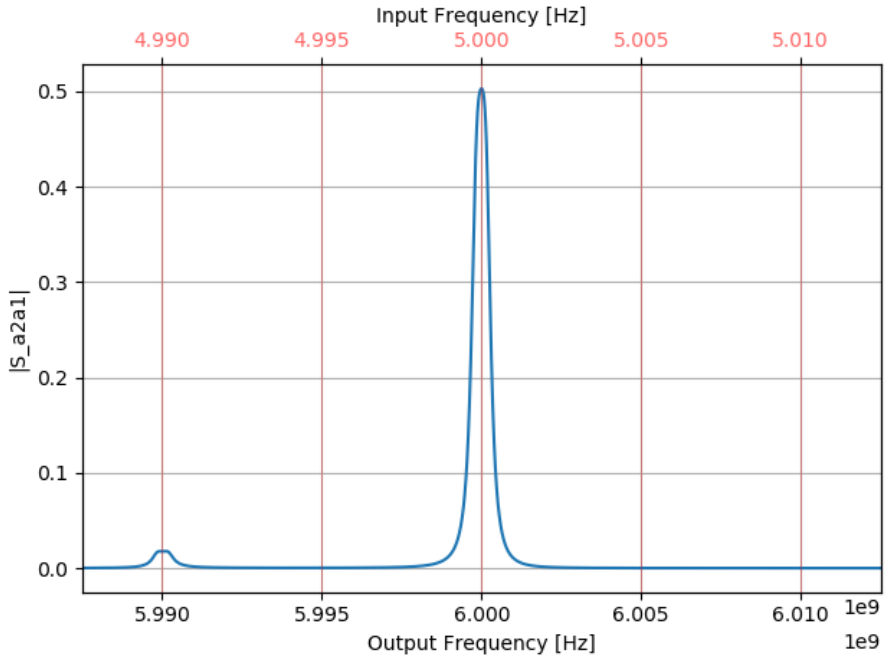

Figure 17: Frequency conversion linear response amplitude

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

This section is dedicated to our perspective for the future of IOpy.

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 be done by hand without approximations. Specially, when the system invloves more oscillating modes including optical cavities, mechanical oscillators or driving fields. Multimode optomechanics which is a growing field in this context can be elevated 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 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. 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 powerfull tones in general it's impossible to apply the same idea and equations of motion can not become time indipendent. In another language, there no longer exists a single rotating frame that every other time variation can be considered slowly variying. 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 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 four-mode isolator, where again each optical cavity is pumped with two drives to be coupled to two different mechanical modes, 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.

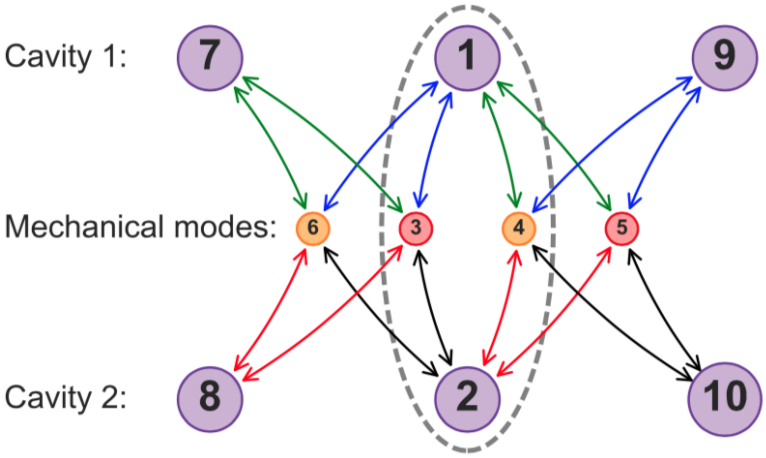

Figure 18: The idea of the expanded mode basis for the four-mode isolator

But in both cases, the calculations rely on truncating an infinite series and using the approximations like rotating wave approximation which we are intended 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 in this way that it injects a coherent signal (a sine wave) to the network and extracts the amplitude and phase from the output of the network. This, we can exactly do with time-domain simulations. In principle we can write the most general Hamiltonian of the optomechanical network as:

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

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

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 $s_{out,jm} = s_{in,jm} - \sqrt{\kappa_{jm}} a_j$. 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. 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 simple cavity. To see the full code go to TimeDomain - SimpleCavity. The results for this simulation are shown in the figures below:

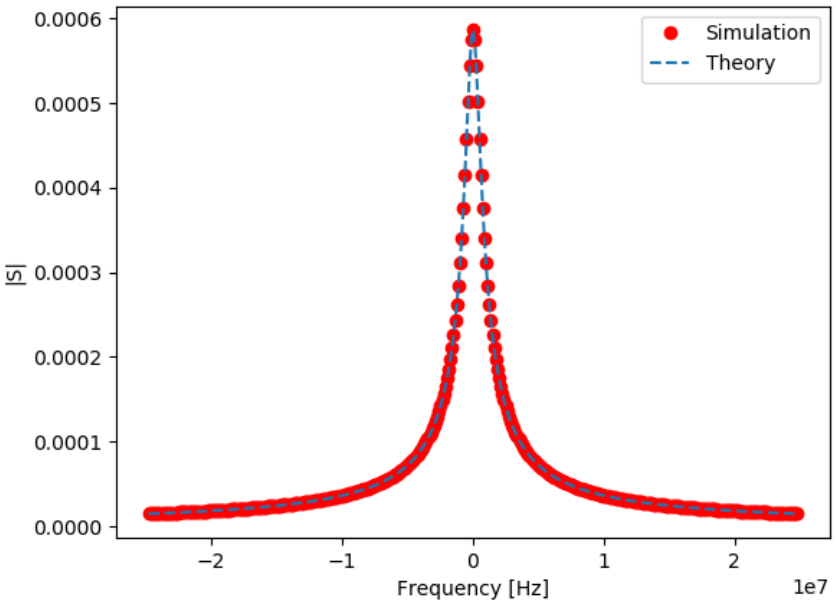

Figure 19: the amplitude of the linear response calculated with time domain simulation

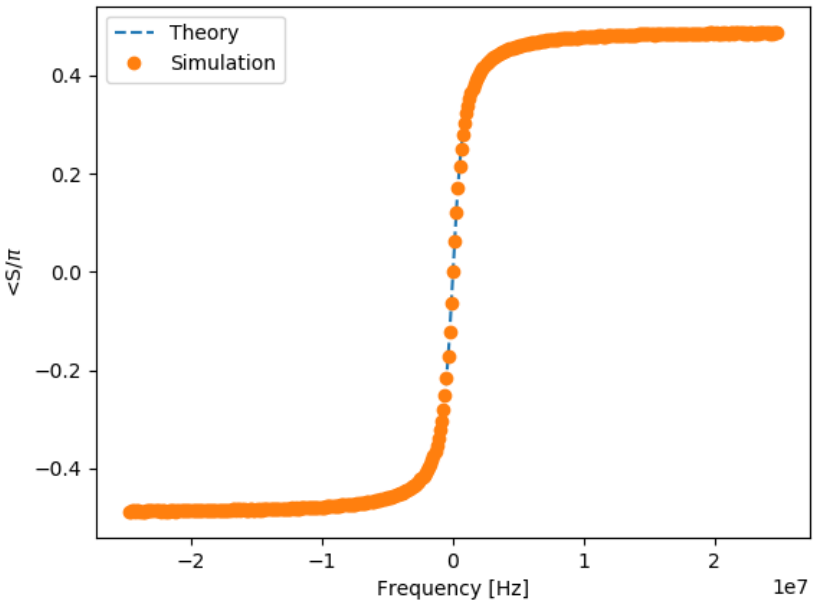

Figure 20: the amplitude of the linear response calculated with time domain simulation

Althogh this idea works for the simple cavity, but for more complex systems like the basic optomechanics, it still requires investment of time as well as time domain simulations skills. Developing a time domian calculation library for the IOpy is one of the main goals of the future for it.

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 of 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, every phenomena which deals with coupled oscillators and can be formulated in input-output formalism can in principle be simulated using it (IO in IOpy stands for Input-Output).

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).