notes

April 30, 2025

1 Using HOM Spacing Measurements to Check Various Thermal Actuator Coupling Factors

1.1 Laying out the problem

We have a slurry of known parameters in the IFO's cold state (no Ring Heaters or self-heating), and with some measurements of the Higher Order Mode spacing (HOM) we can constrain or calculate the values of coupling factors of self-heating and ring heater power to surface curvature changes.

Table of physical parameters we know in the **cold** state:

Parameter	Symbol	Value	Unit
ITMY Radius of Curvature	$R_{\mathrm{i,\,c}}$	1939.2	m
ETMY Radius of Curvature	$R_{\mathrm{e,\,c}}$	2246.9	m
ARM Length	L	3994.47	m

1.2 Cavity g-factors

We can relate the cavity g-factor (a measure of the test masses' curvature relative to the cavity length) to the HOM spacing data.

Here is a list of symbols used in the derivation:

Parameter	Symbol
ITM g-factor	g_i
ITM g-factor, cold-state	$g_{i,c}$
ETM g-factor	g_e
ETM g-factor, cold-state	$g_{e,c}$
Round-trip Gouy Phase	ψ_{rt}
Curvature, cold-state (ITM, ETM)	$D_{i,c}, D_{e,c}$
Curvature, hot-state (ITM, ETM)	D_i, D_e
Self-heating curvature coupling (ITM, ETM)	A_i, A_e
Ring-heater curvature coupling	B
Self-heating power (ITM, ETM)	$P^i_{\mathrm{self}}, P^e_{\mathrm{self}}$
Ring-heater power (ITM, ETM)	$P^i_{\mathrm{rh}}, P^e_{\mathrm{rh}}$

To start we relate the HOM spacing to the cavity g-factor:

$$f_{\mathrm{HOM}} = \frac{\psi_{rt}}{2\pi} f_{\mathrm{FSR}}$$
$$\psi_{rt} = 2\arccos\left(\sqrt{g_i g_e}\right)$$
$$g_i g_e = \cos^2\left(\pi \frac{f_{\mathrm{HOM}}}{f_{\mathrm{FSR}}}\right)$$

Then we express the test mass g-factors in terms of the thermal contributions

$$g_i = 1 - LD_i, \quad D_i = D_{i,c} + A_i P^i_{\mathrm{self}} + BP^i_{\mathrm{rh}}$$
$$g_e = 1 - LD_e, \quad D_e = D_{e,c} + A_e P^e_{\mathrm{self}} + BP^e_{\mathrm{rh}}$$

Multiplying them together and assuming second order terms are small:

$$g_i g_e = 1 - L\left(D_{i,c} + D_{e,c} + A_i P^i_{\mathrm{self}} + A_e P^e_{\mathrm{self}} + B(P^i_{\mathrm{rh}} + P^e_{\mathrm{rh}})\right)$$
$$+L^2\left(D_{i,c}D_{e,c} + D_{i,c}(BP^e_{\mathrm{rh}} + A_e P^e_{\mathrm{self}}) + D_{e,c}A_i P^i_{\mathrm{self}} + \ (A_i A_e + (A_i + A_e)B + B^2 \text{ terms})\right)$$

The terms on the right side are assumed to be small and negligible.

$$g_i g_e = 1 - L(D_{i,c} + D_{e,c}) + L^2 D_{i,c}D_{e,c} - L\left(A_i P^i_{\mathrm{self}} + A_e P^e_{\mathrm{self}} + BP^e_{\mathrm{rh}}\right)$$
$$+L^2\left(D_{i,c}(BP^e_{\mathrm{rh}} + A_e P^e_{\mathrm{self}}) + D_{e,c}A_i P^i_{\mathrm{self}}\right)$$

We can gather terms by groups of cold state terms, B depedent terms, and self-heating terms:

$$g_i g_e = X_c - BLP^e_{\mathrm{rh}}(1 - LD_{i,c}) + X_s = X_c + X_s - BLP^e_{\mathrm{rh}} g_{i,c}$$
$$\cos^2\left(\pi \frac{f_{\mathrm{HOM}}}{f_{\mathrm{FSR}}}\right) = X_c + X_s - BLP^e_{\mathrm{rh}} g_{i,c}$$

where,

$$X_c = 1 - L(D_{i,c} + D_{e,c}) + L^2 D_{i,c}D_{e,c} = g_{i,c}g_{e,c}$$
$$X_s = -L\left(A_i P^i_{\mathrm{self}}(1 - LD_{e,c}) + A_e P^e_{\mathrm{self}}(1 - LD_{i,c})\right) = -L\left(A_i P^i_{\mathrm{self}} g_{i,c} + A_e P^e_{\mathrm{self}} g_{e,c}\right)$$

1.3 Fitting data to recover B and constrain X_s

By fitting a line ($y = mx + b$) to the HOM data as a function of P^e_{rh}, we can recover the HOM change due to self heating as well as the coupling of ring-heater power to curvature (B):

$$\cos^2\left(\pi \frac{f_{\mathrm{HOM}}}{f_{\mathrm{FSR}}}\right) = X_c + X_s - BLP^e_{\mathrm{rh}} g_{i,c}$$
$$y = \cos^2\left(\pi \frac{f_{\mathrm{HOM}}}{f_{\mathrm{FSR}}}\right), \ m = -BLg_{i,c}, \ x = P^e_{\mathrm{rh}}, \ b = X_c + X_s$$

[1]:
```
import numpy as np
import scipy.constants as scc
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

[12]:
```
# known parameters
L = 3994.47 # m
FSR = scc.c / (2 * L) # Hz
static_etm_roc = 2246.90 # m
static_itm_roc = 1939.20 # m
static_etm_curv = 1 / static_etm_roc
static_itm_curv = 1 / static_itm_roc

# measured parameters
rh_power_LG10_phase_data = [
    [1.950, 10.49e3, 10.475e3, 10.5e3, 'init'],
    [1.950, 10.485e3, 10.475e3, 10.49e3, 'end'],
    [2.146, 10.4725e3, 10.46e3, 10.48e3, 'init'],
    [2.146, 10.4620e3, 10.45e3, 10.475e3, 'end'],
    [2.342, 10.448e3, 10.425e3, 10.451e3, 'init'],
    [2.342, 10.4450e3, 10.42e3, 10.46e3, 'end'],
    [2.539, 10.4330e3, 10.40e3, 10.445e3, 'init'],
]
errors = [
            [ abs(rh_power_LG10_phase_data[i][2]/2 -␣
 ↪rh_power_LG10_phase_data[i][1]/2)  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            [abs(rh_power_LG10_phase_data[i][3]/2 -␣
 ↪rh_power_LG10_phase_data[i][1]/2) for i in␣
 ↪range(len(rh_power_LG10_phase_data))]
            ]
# calculate the slope and intercept
def func(X, m, b):
    return m*X + b



fig, axs = plt.subplots(figsize=(12,9))
color = {'init': 'C0', 'end': 'C1'}
thickness = {'init': 2, 'end': 1}
axs.scatter([rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            [rh_power_LG10_phase_data[i][1]/2  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            label='Data',
            c=[color[rh_power_LG10_phase_data[i][-1]]  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],)
```

```
axs.errorbar([rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            [rh_power_LG10_phase_data[i][1]/2  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            yerr=errors,
            fmt='none',
            ecolor=[color[rh_power_LG10_phase_data[i][-1]]  for i in␣
 ↪range(len(rh_power_LG10_phase_data))],
            ls='',
            elinewidth=[thickness[rh_power_LG10_phase_data[i][-1]]  for i in␣
 ↪range(len(rh_power_LG10_phase_data))]
            )


#   initial phase  #

# upper error fit
X_c = (1 - L * static_itm_curv)*(1 - L * static_etm_curv)
g_ic = 1 - L * static_itm_curv
rh_linspace = np.linspace(rh_power_LG10_phase_data[0][0],␣
 ↪rh_power_LG10_phase_data[-1][0], 100)

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][3] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init'], gige)
slope, intercept = popt
Bupper = slope / (-L * g_ic)
X_s_upper = intercept - X_c
self_heating_curv_upper = -X_s_upper/L
upperbound = FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/ np.pi
HOM_noRH_upper = FSR*np.arccos(np.sqrt(intercept))/np.pi

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][2] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init'], gige)
slope, intercept = popt
Blower = slope / (-L * g_ic)
X_s_lower = intercept - X_c
self_heating_curv_lower = -X_s_lower/L
lowerbound = FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/ np.pi
```

```
HOM_noRH_lower = FSR*np.arccos(np.sqrt(intercept))/np.pi

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][1] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'init'], gige)
slope, intercept = popt
Bmean = slope / (-L * g_ic)
X_s_mean = intercept - X_c
self_heating_curv_mean = -X_s_mean/L
HOM_noRH = FSR*np.arccos(np.sqrt(intercept))/np.pi


HOM_cold = FSR*np.arccos(np.sqrt(X_c))/np.pi
label = "Fit:\n" + f"    B = {Bmean:.3e} +/- {np.
 ↪mean([Bmean-Bupper,Blower-Bmean]):.2e}[D/W]\n    " + \
       r"$_\text{HOM, no rh}$" + \
       f"={HOM_noRH:.1f} +/- {np.
 ↪mean([HOM_noRH_lower-HOM_noRH,HOM_noRH-HOM_noRH_upper]):.1f}[Hz]\n"
axs.plot(np.linspace(rh_power_LG10_phase_data[0][0],␣
 ↪rh_power_LG10_phase_data[-1][0], 100),
        FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/np.pi,
        ls = '--', color = 'C0',
        label=label)
axs.fill_between(rh_linspace, lowerbound, upperbound,
                 color='C0', alpha=.2)


#   end phase  #

# upper error fit
X_c = (1 - L * static_itm_curv)*(1 - L * static_etm_curv)
g_ic = 1 - L * static_itm_curv
g_ec = 1 - L * static_etm_curv
rh_linspace = np.linspace(rh_power_LG10_phase_data[0][0],␣
 ↪rh_power_LG10_phase_data[-1][0], 100)

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][3] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end'], gige)
slope, intercept = popt
```

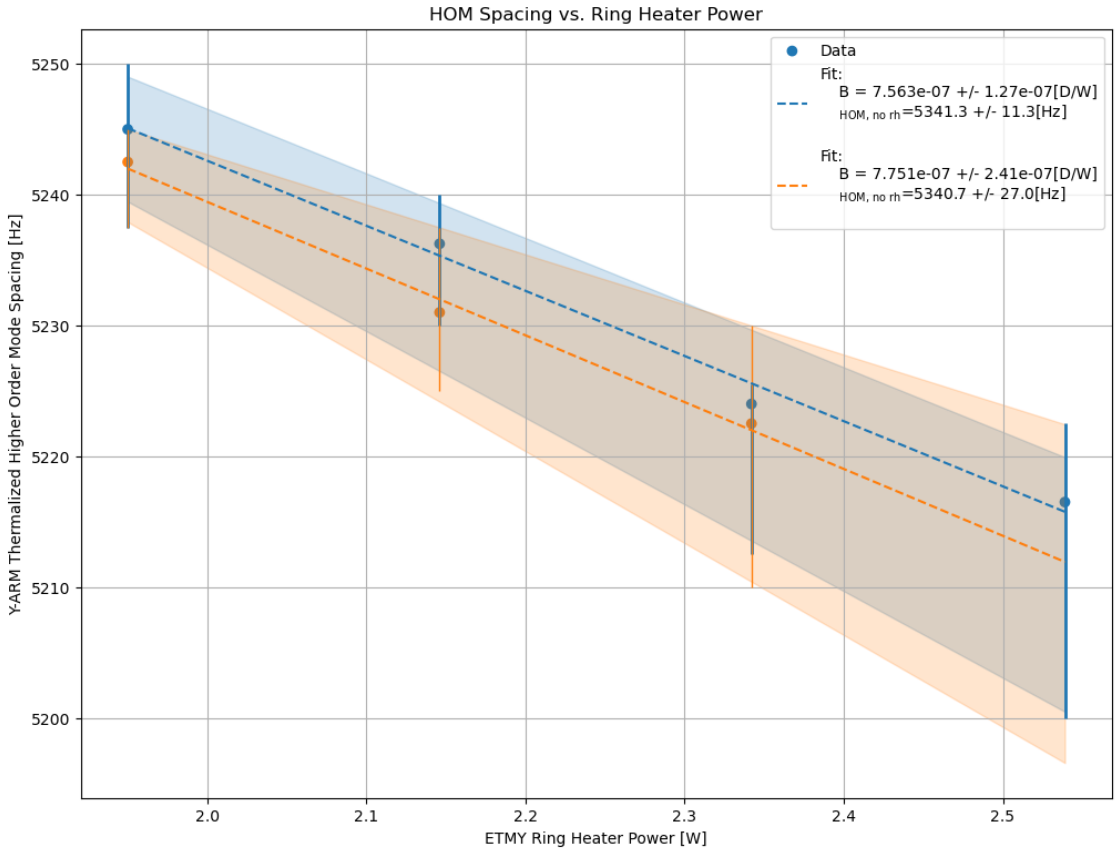
```
Bupper = slope / (-L * g_ic)
X_s_upper = intercept - X_c
self_heating_curv_upper = -X_s_upper/L
upperbound = FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/ np.pi
HOM_noRH_upper = FSR*np.arccos(np.sqrt(intercept))/np.pi

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][2] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end'], gige)
slope, intercept = popt
Blower = slope / (-L * g_ic)
X_s_lower = intercept - X_c
self_heating_curv_lower = -X_s_lower/L
lowerbound = FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/ np.pi
HOM_noRH_lower = FSR*np.arccos(np.sqrt(intercept))/np.pi

gige = [np.cos(np.pi * rh_power_LG10_phase_data[i][1] / 2 / FSR)**2 for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end']
popt, pcov = curve_fit(func, [rh_power_LG10_phase_data[i][0]  for i in␣
 ↪range(len(rh_power_LG10_phase_data)) if rh_power_LG10_phase_data[i][-1] ==␣
 ↪'end'], gige)
slope, intercept = popt
Bmean = slope / (-L * g_ic)
X_s_mean = intercept - X_c
self_heating_curv_mean = -X_s_mean/L
HOM_noRH = FSR*np.arccos(np.sqrt(intercept))/np.pi

HOM_cold = FSR*np.arccos(np.sqrt(X_c))/np.pi
label = "Fit:\n" + f"    B = {Bmean:.3e} +/- {np.
 ↪mean([Bmean-Bupper,Blower-Bmean]):.2e}[D/W]\n    " + \
       r"$_\text{HOM, no rh}$" + \
       f"={HOM_noRH:.1f} +/- {np.
 ↪mean([HOM_noRH_lower-HOM_noRH,HOM_noRH-HOM_noRH_upper]):.1f}[Hz]\n"
axs.plot(np.linspace(rh_power_LG10_phase_data[0][0],␣
 ↪rh_power_LG10_phase_data[-1][0], 100),
         FSR*np.arccos(np.sqrt(func(rh_linspace, *popt)))/np.pi,
         ls = '--', color = 'C1',
         label=label)
axs.fill_between(rh_linspace, lowerbound, upperbound,
                 color='C1', alpha=.2)
```

```
axs.set_xlabel("ETMY Ring Heater Power [W]")
axs.set_ylabel("Y-ARM Thermalized Higher Order Mode Spacing [Hz]")
axs.grid(True, 'both', 'both')
axs.set_title("HOM Spacing vs. Ring Heater Power")
axs.legend()
fig.savefig("./figures/rh_power_to_surf_curv_fit.pdf")
```



2 Using TCS SIM value to project its estimate of HOM spacing shift due to self heating.

[21]:
```
print(f"OMC Data estimate of HOM spacing shift due to self-heating alone:
  ↪\n\t{HOM_noRH - HOM_cold:.1f} Hz", end="")
print(f", from {HOM_cold:.1f} Hz --> {HOM_noRH:.1f} Hz")

tcs_sim_itmy_abs_pwr = 0.14431 # W
tcs_sim_itmy_self_surf_dfs_gain = -3.652e-05 # D/W
tcs_sim_etmy_abs_pwr = 0.0808072 #W
tcs_sim_etmy_self_surf_dfs_gain = -2.931e-05 # D/W
```

```
tcs_sim_Xs = -L*(tcs_sim_itmy_abs_pwr*tcs_sim_itmy_self_surf_dfs_gain*g_ic +␣
 ↪tcs_sim_etmy_abs_pwr*tcs_sim_etmy_self_surf_dfs_gain*g_ec)
tcs_sim_HOM_noRH = FSR * np.arccos(np.sqrt(X_c + tcs_sim_Xs)) / np.pi
print(f"\nTCS SIM estimate of HOM spacing shift due to self-heating alone:
 ↪\n\t{tcs_sim_HOM_noRH - HOM_cold:.1f} Hz", end="")
print(f", from {HOM_cold:.1f} Hz --> {tcs_sim_HOM_noRH:.1f} Hz")

print(f"\nDifference Factor : TCS_SIM / OMC_DATA = {(tcs_sim_HOM_noRH -␣
 ↪HOM_cold) / (HOM_noRH - HOM_cold):.2f}")
```

```
OMC Data estimate of HOM spacing shift due to self-heating alone:
        174.6 Hz, from 5166.1 Hz --> 5340.7 Hz

TCS SIM estimate of HOM spacing shift due to self-heating alone:
        451.5 Hz, from 5166.1 Hz --> 5617.7 Hz

Difference Factor : TCS_SIM / OMC_DATA = 2.59
```