pygwinc:
A roadmap to a generalized noise budgeting tool for opto-mechanical experiments

Jameson Graef Rollins
Christopher Wipf

IFO Simulations call
February 26, 2019

pygwinc

pygwinc is a pure-python port of the MATLAB-based Gravitational Wave Interferometer Noise Calculator (GWINC):

https://git.ligo.org/gwinc/pygwinc

(Old MATLAB GWINC ("matgwinc") has also moved to git:)

https://git.ligo.org/gwinc/matgwinc

pygwinc

- Collection of mostly analytic noise calculations for LIGO-like interferometers.
- IFO "models" stored in YAML files.
- $\times$1000 faster than MATLAB GWINC.
- Command line interface for plotting budgets from `IFO.yaml` description files.
- Uses inspiral-range package for calculating range figures of merit.
- Can use MATLAB python engine for calculation/comparison with old MATLAB-based GWINC.

pygwinc YAML IFO model descriptions

Simple, readable, extensible, common structured data format for storing experimental parameters.

`aLIGO.yaml`:

```yaml
Infrastructure:
  Length: 3995.0
  ResidualGas:
    mass: 3.35e-27
    polarizability: 7.8e-31
    pressure: 4.0e-07
  Temp: 290.0
Laser:
  Power: 125.0
  Wavelength: 1.064e-06
Materials:
  Coating:
    Alphahighn: 3.6e-06
    Alphalown: 5.1e-07
    Betahighn: 1.4e-05
    Betalown: 8.0e-06
    CVhighn: 2100000.0
    CVlown: 1641200.0
    Indexhighn: 2.06539
    Indexlown: 1.45
    Phihighn: 0.00036
    Phihighn_slope: 0.1
    Philown: 5.0e-05
```

pygwinc usage

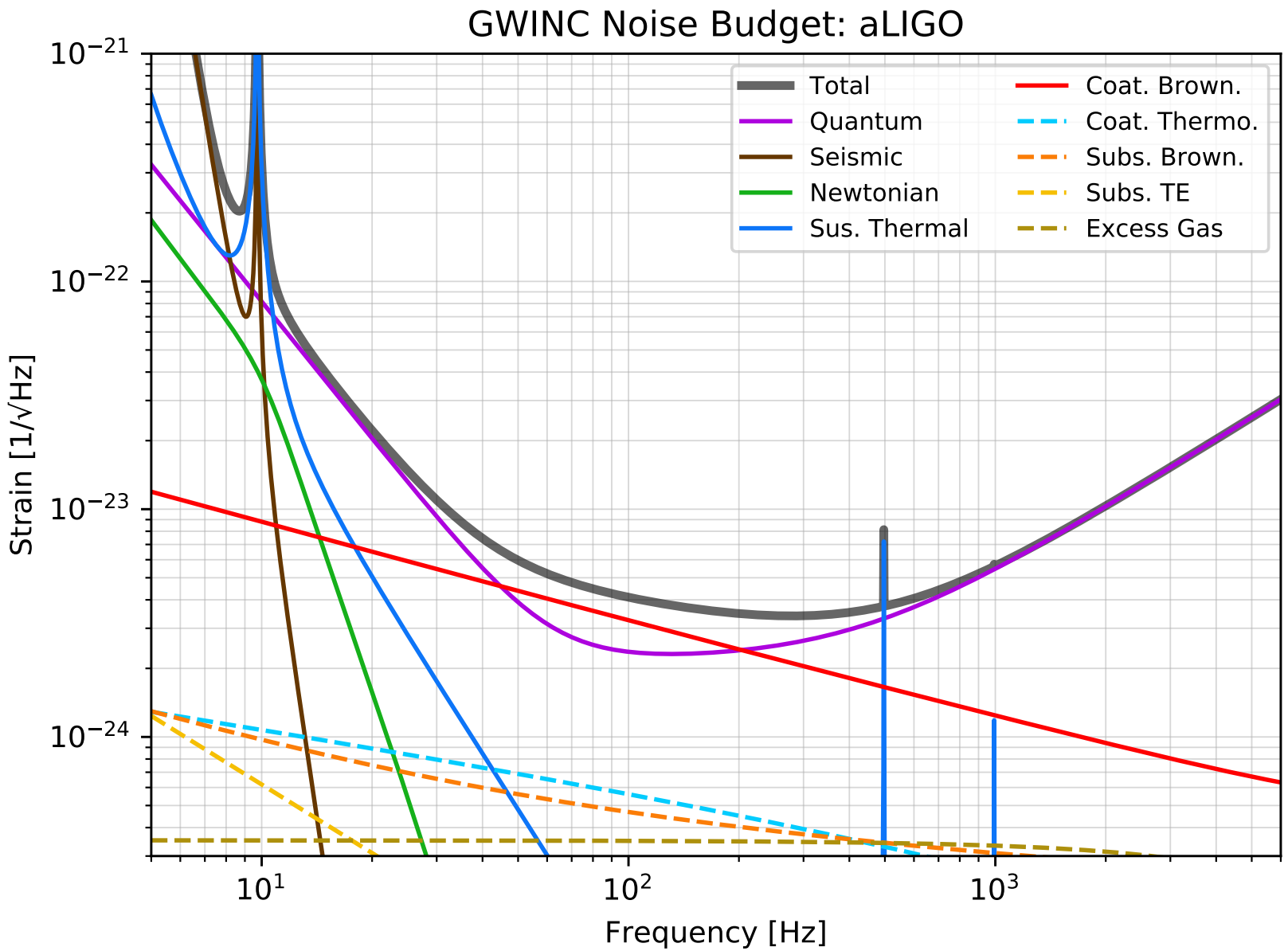
Load and plot budgets programmatically:

```
>>> import gwinc
>>> import numpy as np
>>> freq = np.logspace(1, 3, 1000)
>>> ifo = gwinc.load_ifo('aLIGO')
>>> ifo = gwinc.precompIFO(freq, ifo)
>>> noises = gwinc.noise_calc(freq, ifo)
>>> gwinc.plot_noise(ifo, noises)
```

or from the command line:

```
$ gwinc aLIGO
```

pygwinc noise budget plot



pygwinc development

Initial motivation:

- Python.
- Better code management and quality control.
- Improve performance.

Semi-self-imposed constraints:

- Make it looks as much like matgwinc internally as possible, to ease transition for developers.
- Retain backwards compatibility with matgwinc (ability to load and execute existing matgwinc `ifo.mat` files).
- Strict validation with matgwinc.

pygwinc noise calculators

Basic noise function:

```
gwinc.noise.quantum.shotrad
gwinc.noise.seismic.seismic
gwinc.noise.newtonian.gravg
gwinc.noise.suspensionthermal.susptherm
gwinc.noise.coatingthermal.coatbrownian
gwinc.noise.coatingthermal.thermooptic
gwinc.noise.substratethermal.subbrownian
gwinc.noise.substratethermal.subtherm
gwinc.noise.residualgas.gas
```

More being added for Voyager, CE...

pygwinc noise calculators

From `gwinc.noise.coatingthermal`:

```
def coatbrownian(f, ifo):
    """Optical coating Brownian thermal noise"""

    Length = ifo.Infrastructure.Length
    wBeam_ITM = ifo.Optics.ITM.BeamRadius
    wBeam_ETM = ifo.Optics.ETM.BeamRadius
    dOpt_ITM = ifo.Optics.ITM.CoatLayerOpticalThickness
    dOpt_ETM = ifo.Optics.ETM.CoatLayerOpticalThickness

    # compute Brownian noise for specified coating structure
    SbrITM  = getCoatBrownian(f, ifo, wBeam_ITM, dOpt_ITM)
    SbrETM  = getCoatBrownian(f, ifo, wBeam_ETM, dOpt_ETM)

    n = 2 * (SbrITM + SbrETM) * ifo.gwinc.dhdl_sqr

    return n
```

Note very similar look to matgwinc (including MATLAB `struct`-like data structure for IFO parameters).

pygwinc limitations

GWINC assumes a LIGO-like detector configuration, i.e. suspended Michelson interferometer.

- Prevents encoded knowledge of the fundamental noise sources from being used in other contexts, experimental typologies, etc.
- Prevents use of more sophisticated tools for calculating noise transfer functions (e.g. Finesse, Optickle, etc.)

Can we separate out the fundamental noise calculations from the transfer functions/calibrations that transform them to the desired measurement basis?

common noise budget patterns

Meanwhile, we spend a lot of time working on general experimental noise budgets (for LIGO, 40m, table-top experiments, etc.)

Most noise budgets follow similar patterns:

- Combination of measured and analytically-derived noise terms.
- Separation of noise sources and calibrations from sources to measurement port.
- Common functions for loading data, calculating spectra, etc.
- Standard plot styles.

Specifically for LIGO we want to re-write the LIGO (H1/L1) noise budgets in python, share common code between sites, export to nbweb, etc.

a common noise budget tool?

If we need the same thing in pygwinc that we want for noise budgeting in general, can pygwinc not just provide a generalized noise budgeting tool? Possible roadmap:

- Add generalized `BudgetItem` class for defining noise terms and calibrations that can e.g. load static data, calculate noise spectra, return PSD at specified frequency points.
- Separate fundamental noise calculators from assumptions about experiment topology.
- Budgets become python module/package that defines `Noise` and `Calibration` classes, and a .yaml file that stores experimental parameters.

What follows is PRELIMINAY concept outline/prototype...

gwinc.nb.Noise class with analytic calculation

```
from gwinc import nb

class CoatingBrownian(nb.Noise):
    """Coating Brownian

    """
    # attribute for plotting style arguments
    style = dict(
        label='Coat. Brown.',
        color='#fe0002',
    )

    # method to calculate/return noise PSD
    def calc(self):
        self.ifo = gwinc.precompIFO(self.freq, self.ifo)
        return noise.coatingthermal.coatbrownian(
            self.freq,
            self.ifo,
        )
```

gwinc.nb.Noise for measured data

```
from gwinc import nb

class MICH(nb.Noise):
    """MICH Controls Noise

    """
    style = dict(
        label='MICH',
        color=[0.20, 0.10, 1.00],
    )

    def calc(self):
        # excess power projection from witness to DARM
        # from broadband noise excitation
        f_meas, S_proj = excess_power_projection_from_dtt(
            lpath('couplings/MICH_excitation.xml'),
            'H1:LSC-MICH_OUT_DQ(REF7)',
            'H1:LSC-MICH_OUT_DQ',
            'H1:CAL-DELTAL_EXTERNAL_DQ(REF6)',
            'H1:CAL-DELTAL_EXTERNAL_DQ',
        )
        return self.interpolate(f_meas, S_proj)
```

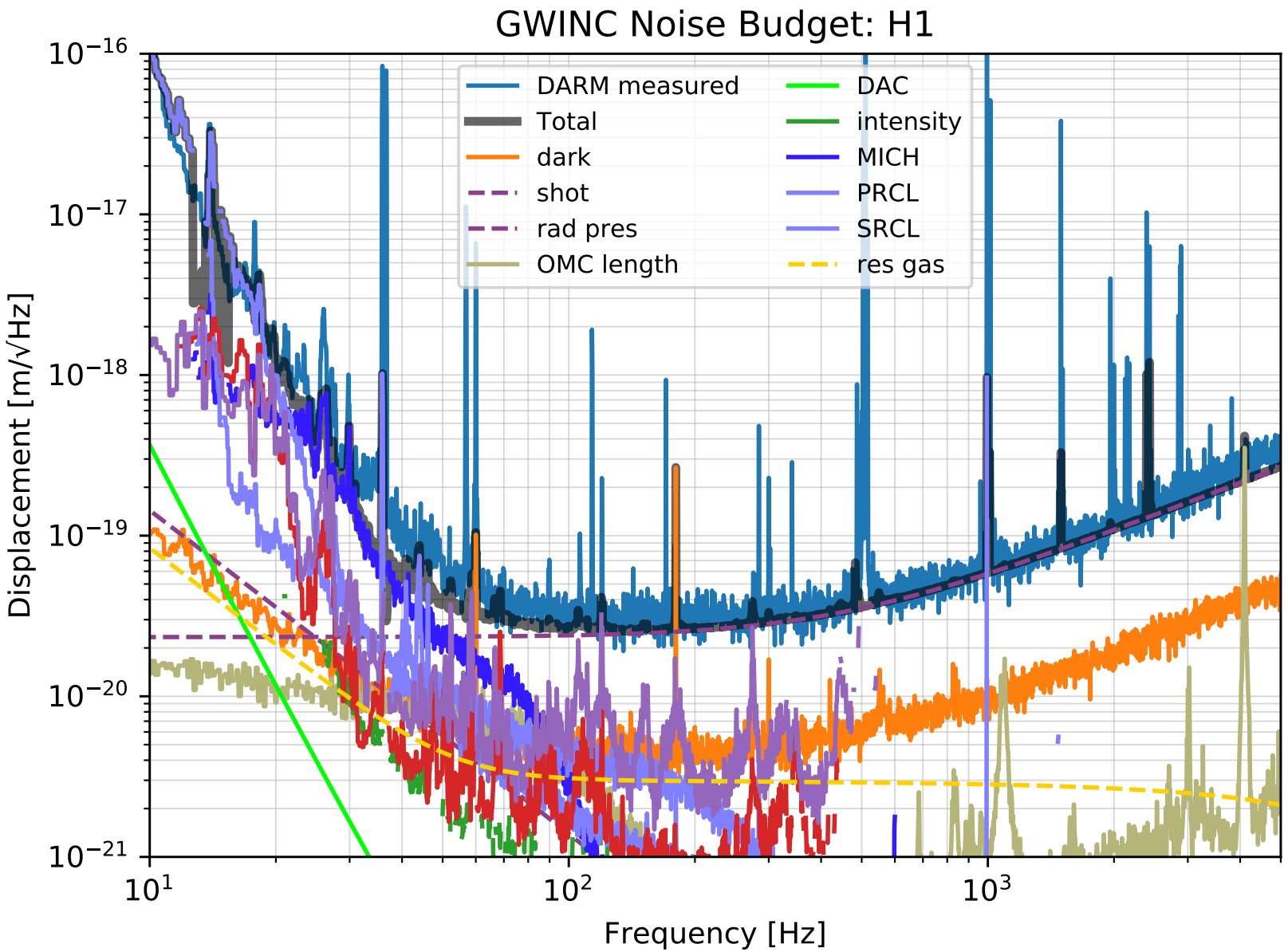
gwinc.nb for measured noise with calibration

```
from gwinc import nb

class ClosedLoopSensing(nb.Calibration):
    def calc(self):
        ...
        return cal

class Shot(nb.Noise):
    """AS PD shot noise

    """
    # calibration to be applied to PSD from self.calc()
    calibration = ClosedLoopSensing

    # NDS channel data to be fetched
    nds_channels = [
        'H1:OMC-DCPD_SUM_OUT16',
    ]

    def calc(self):
        data = self.nds_data['H1:OMC-DCPD_SUM_OUT16']
        ...
        return psd
```

executing gwinc.nb

```
>>> import numpy as np
>>> import gwinc
>>> Budget = gwinc.load_budget('/path/to/mybudget.py')
>>> freq = np.logspace(1, 3, 1000)
>>> budget = Budget(freq)
>>> budget.execute()
>>> gwinc.plot_noise(budget)
```

pygwinc H1 noise budget



issues and plans

Have working prototype, but still very preliminary. More work, code cleanup, refinement, etc. needed before official release.

Nonetheless, looking for feedback/thoughts on this approach.

Considerations:

- Want to keep everything as conceptually simple and flexible as possible. Don't overly encode or enforce behavior.
- Quantum noise is tricky, since it's deeply embedded into the overall experiment topology.
- Add optimizations for "online" noise budgets.
- Might need to break backwards compatibility with matgwinc.