Week 1

Intro

Project aims to use MCMC algorithms in combination with FINESSE models to assess the sensitivity of various ifo parameters.

Eventually, this would be compared to thermal steady state data to create the MCMC likelihood function.

This week I just added noise to simulated parameter values to use as the ground truth values.

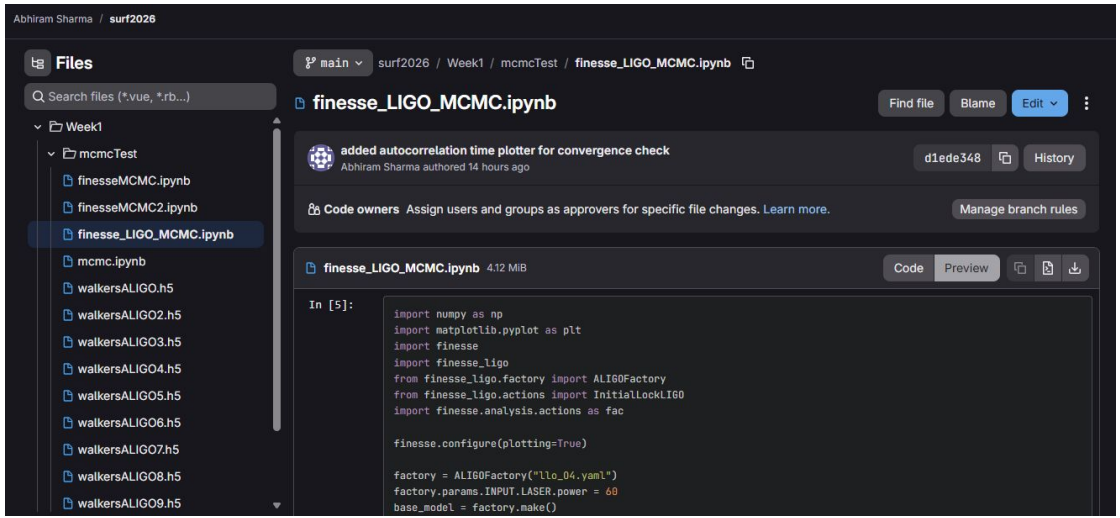
Picking Algorithm/Package

Decided to use the emcee python package which implements an Affine Invariant MCMC Ensemble Sampler which I heard was used by others in LIGO.

I also tried PyMC which is good for user–defined custom samplers and dynesty which uses Dynamic Nested Sampling and is good for multimodal distributions.

Main Test Code

After running smaller test notebooks with finesse and the MCMC packages separately, I started working on a notebook to do parameter recovery and I uploaded it to a repo under my git-ligo account.



Main Test Code - Cell 1

First I made a factory ligo model from finesse-ligo and added a subset of the labeled detectors from the run.py file. I used the vector of detector values, with gaussian noise added at 2.5 % of the output magnitude, as the MCMC objective. To start, I only used 11 detector outputs but in Week 2 I added the rest.

```
base_model.parse("""
    fd E_refl_u45 M5.p2.o f=+f2
    fd E_refl_l45 M5.p2.o f=-f2
    fd E_refl_c0 M5.p2.o f=0

    fd E_prc_u45 PRM.p1.o f=+f2
    fd E_prc_l45 PRM.p1.o f=-f2
    fd E_prc_c0 PRM.p1.o f=0

    fd E_xarm_c0 ETMX.p1.o f=0
    fd E_yarm_c0 ETMY.p1.o f=0

    pd TRX ETMX.p2.o
    pd TRY ETMY.p2.o
    pd OMC_DCPD OMC_OC.p2.o
""")
```

```
    np.random.seed(42)
    for key in target_keys:
        val = out_true[key]
        if np.iscomplexobj(val):
            val = np.abs(val)

        err = 0.025 * val + 1e-9
        obs = val + err * np.random.randn()

        y_obs[key] = obs
        y_err[key] = err
        print(f"{key}: {obs} (err: {err})")
```

✓ 7.2s

```
True ITMX ROC: 1937.900 m
True ETMX ROC: 2240.000 m
True Laser Power: 60.000 W
OMC_DCPD: 0.025613825330485004 (err: 0.0006324
TRX: 2.365586379067794 (err: 0.059344792132001
TRY: 2.5816523329066 (err: 0.0635128949703183)
E_refl_c0: [1.43657722e-01 1.52302986e-09 1.52
 1.52302986e-09 2.73559332e-03] (err: [3.45971
 1.00000000e-09 6.58823146e-05])
```

Main Test Code - Cell 2

Then I defined the log probability functions needed for emcee with physical cutoffs for proposed parameter values

```
def log_prior(theta):
    rc_itm, rc_etm, rc_laser = theta
    if (1918.0 < rc_itm < 1958.0) and (2225.0 < rc_etm < 2265.0) and (50 < rc_laser < 70):
        return 0.0
    return -np.inf

VERBOSE = False

def log_likelihood(theta, obs_dict, err_dict):
    rc_itm, rc_etm, rc_laser = theta
    global base_model
    global warm_model_cache

    is_warm_start = False
    nearest_model_template = None

    if len(warm_model_cache) > 0:
        distances = [np.linalg.norm(np.array(theta) - np.array(c['theta'])) for c in warm_model_cache]
        min_idx = np.argmin(distances)

        nearest_model_template = warm_model_cache[min_idx]['model']

    if nearest_model_template is not None:
        model = nearest_model_template #.deepcopy()
        active_lock = fac.RunLocks(max_iterations=50, exception_on_fail=True)
```

Main Test Code - Cell 2

Most of the complexity with using this MCMC package will be in defining the log_likelihood function. Right now, I do not think the code is parallelizable because I tried to speed up the single core execution by modifying a single finesse model's parameters rather than using deepcopy which lets it run about 30~40% faster.

```python
def log_likelihood(theta, obs_dict, err_dict):
    rc_itm, rc_etm, rc_laser = theta
    global base_model
    global warm_model_cache

    is_warm_start = False
    nearest_model_template = None

    if len(warm_model_cache) > 0:
        distances = [np.linalg.norm(np.array(theta) - np.array(c['theta'])) for c in warm_model_cache]
        min_idx = np.argmin(distances)

        nearest_model_template = warm_model_cache[min_idx]['model']

    if nearest_model_template is not None:
        model = nearest_model_template #.deepcopy()
        active_lock = fac.RunLocks(max_iterations=50, exception_on_fail=True)
```

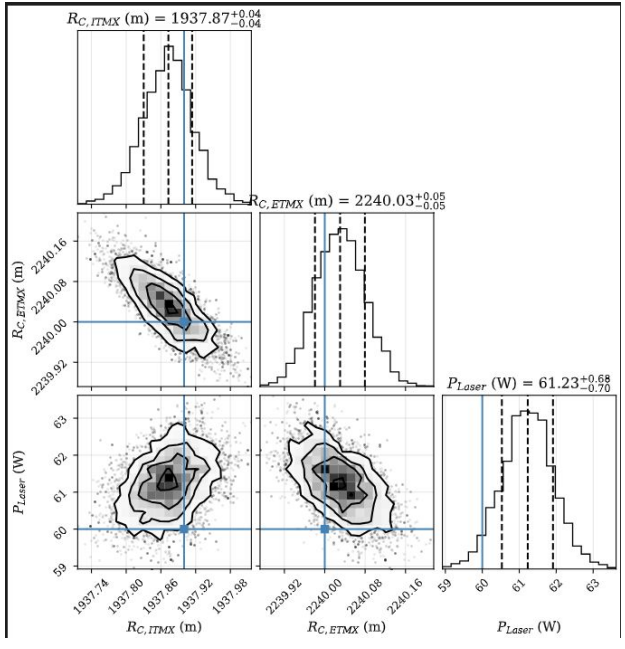
Main Test Code - Cell 3

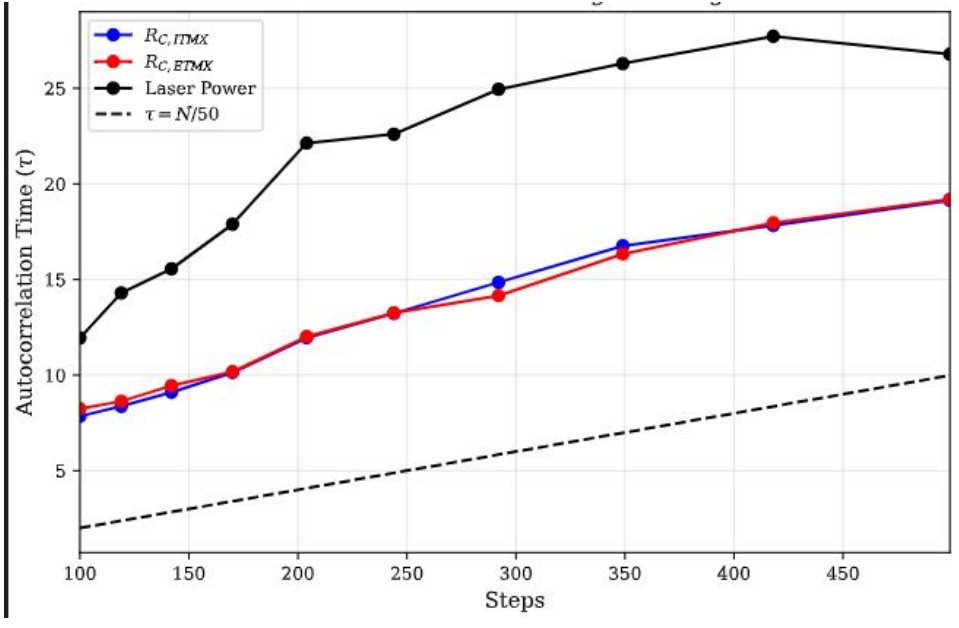
This cell defines the sampling parameters like dimension of parameter space to sample, number of walker in ensemble, etc. The sampling is saved to a h5 file as it progresses so it can be paused and resumed at any point. I also make sure that the initial guesses for the walkers start in a valid, locked ifo state.

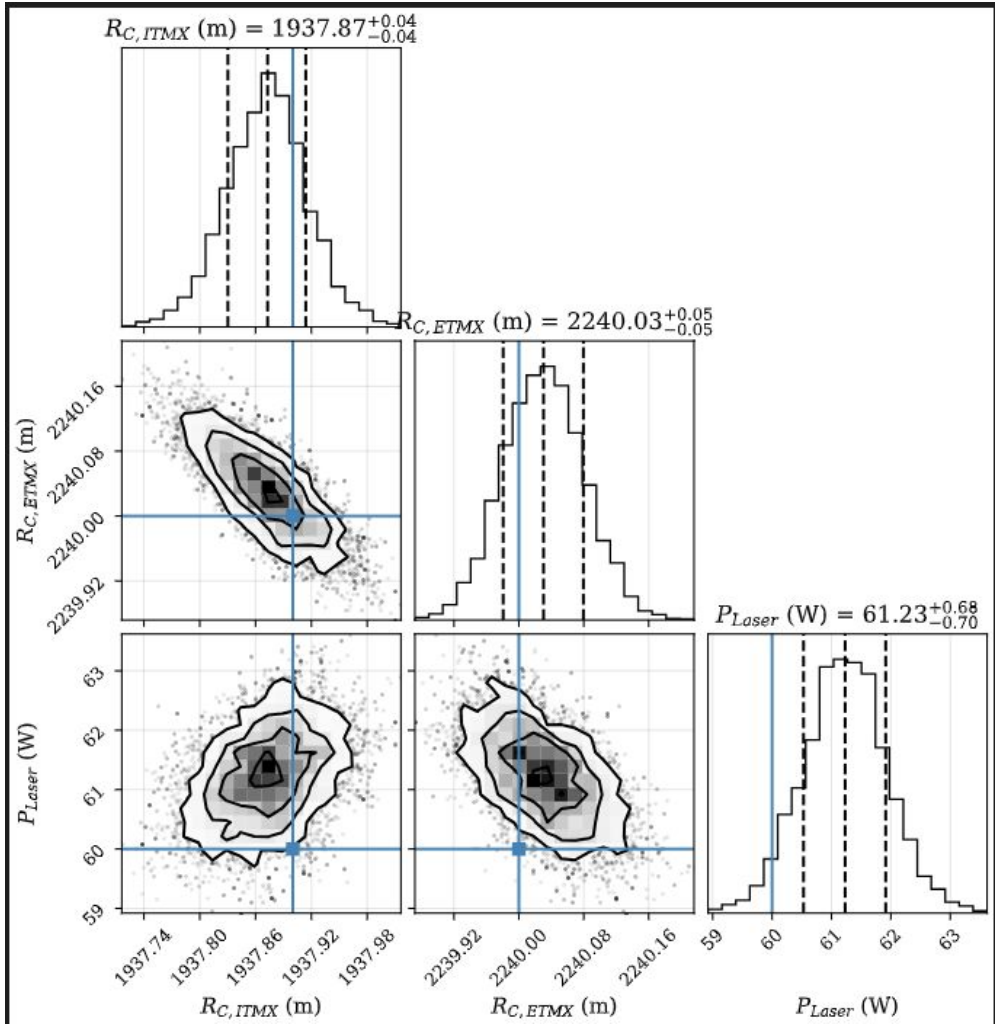
```
ndim = 3
nwalkers = 16
max_steps = 500
chunk_size = 100


initial_guess = np.array([true_itmx_roc, true_etmx_roc, true_laser_power])

filename = "walkersALIGO10.h5"

output_dir = os.path.dirname(filename)
if output_dir:
    os.makedirs(output_dir, exist_ok=True)

backend = emcee.backends.HDFBackend(filename)

if os.path.exists(filename) and backend.iteration > 0:
    print(f"Resuming existing simulation from step {backend.iteration}")
    p0 = None
else:
    print("Starting new simulation")
    backend.reset(nwalkers, ndim)
    p0 = np.zeros((nwalkers, ndim))
    np.random.seed(42)

    for i in range(nwalkers):
        is_valid = False
        attempts = 0
        while not is_valid:
            attempts += 1
```

Main Test Code - Cell 4, 5

Cell 4 and 5 just makes the corner plot from the h5 file and also plots the autocorrelation time for the parameters since that is how the authors of emcee recommended to check for distribution convergence.






$R_{C,ITMX}$ (m) = $1937.87^{+0.04}_{-0.04}$
$R_{C,ETMX}$ (m) = $2240.03^{+0.05}_{-0.05}$
P_{Laser} (W) = $61.23^{+0.68}_{-0.70}$
$R_{C,ETMX}$ (m)
2240.16
2240.08
2240.00
2239.92
P_{Laser} (W)
63
62
61
60
59
1937.74
1937.80
1937.86
1937.92
1937.98
$R_{C,ITMX}$ (m)
2239.92
2240.00
2240.08
2240.16
$R_{C,ETMX}$ (m)
59
60
61
62
63
P_{Laser} (W)

Modifying code from commented (running models separately) to uncommented lines (running models in series) also speeds up the sampler by about 2x

```
lock = InitialLockLIGO(
    exception_on_lock_fail=False,
    lock_steps=100,
    gain_scale=0.4,
    pseudo_lock_arms=False,
    run_locks=True,
)
# base_model.run(lock)
# out_true = base_model.run()
main_action = fac.Series(lock, fac.Noxaxis(name="measurement"))
out_true = base_model.run(main_action)
```

```
val = out_true["measurement"][key]
# val = out_true[key]
```

```python
def log_likelihood(theta, obs_dict, err_dict):
    # print(f"THETA = {theta}")
    rc_itm, rc_etm = theta
    global base_model

    model = base_model

    active_lock = fac.RunLocks(max_iterations=40, exception_on_fail=True)
    main_action = fac.Series(active_lock, fac.Noxaxis(name="measurement"))

    model.ITMX.Rcx = rc_itm
    model.ITMX.Rcy = rc_itm
    model.ETMX.Rcx = rc_etm
    model.ETMX.Rcy = rc_etm
    # print(f"  After setting: ITMX.Rcx={model.ITMX.Rcx.value}, ETMX.Rcx={model.ETMX.Rcx.value}")

    try:
        out_series = model.run(main_action)

    except LostLock:
        fallback_lock = InitialLockLIGO(
            exception_on_lock_fail=True,
            lock_steps=100,
            gain_scale=0.4,
            pseudo_lock_arms=False,
            run_locks=True
        )
        fallback_action = fac.Series(fallback_lock, fac.Noxaxis(name="measurement"))
        try:
            out_series = model.run(fallback_action)
        except Exception:
            return -np.inf

    out = out_series["measurement"]
    #print(out)
    chi2 = 0.0
```