Inference on the Champagne Model using a Gaussian Process

TODO

- Change outputs

Setting up the Champagne Model

Imports

```
import pandas as pd
import numpy as np
from typing import Any
import matplotlib.pyplot as plt

from scipy.stats import qmc
from scipy.stats import norm

import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow_probability.python.distributions import normal

tfb = tfp.bijectors
tfd = tfp.distributions
tfk = tfp.math.psd_kernels
tfp_acq = tfp.experimental.bayesopt.acquisition
```

```
2024-04-29 09:00:24.065109: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensor
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-04-29 09:00:24.696057: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT W
```

Model itself

```
np.random.seed(590154)

population = 1000
initial_infecteds = 10
epidemic_length = 1000
number_of_events = 15000

pv_champ_alpha = 0.4  # prop of effective care
pv_champ_beta = 0.4  # prop of radical cure
pv_champ_gamma_L = 1 / 223  # liver stage clearance rate
pv_champ_delta = 0.05  # prop of imported cases
pv_champ_lambda = 0.04  # transmission rate
pv_champ_f = 1 / 72  # relapse frequency
pv_champ_r = 1 / 60  # blood stage clearance rate


def champagne_stochastic(
    alpha_,
    beta_,
    gamma_L,
    lambda_,
    f,
    r,
    N=population,
    I_L=initial_infecteds,
    I_0=0,
    S_L=0,
    delta_=0,
    end_time=epidemic_length,
    num_events=number_of_events,
):
    if (0 > (alpha_ or beta_)) or (1 < (alpha_ or beta_)):
        return "Alpha or Beta out of bounds"
    if 0 > (gamma_L or lambda_ or f or r):
        return "Gamma, lambda, f or r out of bounds"
```

```python
t = 0
S_0 = N - I_L - I_0 - S_L
inc_counter = 0

list_of_outcomes = [
    {"t": 0, "S_0": S_0, "S_L": S_L, "I_0": I_0, "I_L": I_L, "inc_counter": 0}
]

prop_new = alpha_ * beta_ * f / (alpha_ * beta_ * f + gamma_L)
i = 0

while (i < num_events) or (t < 30):
    i += 1
    if S_0 == N:
        while t < 31:
            t += 1
            new_stages = {
                "t": t,
                "S_0": N,
                "S_L": 0,
                "I_0": 0,
                "I_L": 0,
                "inc_counter": inc_counter,
            }
            list_of_outcomes.append(new_stages)
        break

    S_0_to_I_L = (1 - alpha_) * lambda_ * (I_L + I_0) / N * S_0
    S_0_to_S_L = alpha_ * (1 - beta_) * lambda_ * (I_0 + I_L) / N * S_0
    I_0_to_S_0 = r * I_0 / N
    I_0_to_I_L = lambda_ * (I_L + I_0) / N * I_0
    I_L_to_I_0 = gamma_L * I_L
    I_L_to_S_L = r * I_L
    S_L_to_S_0 = (gamma_L + (f + lambda_ * (I_0 + I_L) / N) * alpha_ * beta_) * S_L
    S_L_to_I_L = (f + lambda_ * (I_0 + I_L) / N) * (1 - alpha_) * S_L

    total_rate = (
        S_0_to_I_L
        + S_0_to_S_L
        + I_0_to_S_0
        + I_0_to_I_L
        + I_L_to_I_0
```

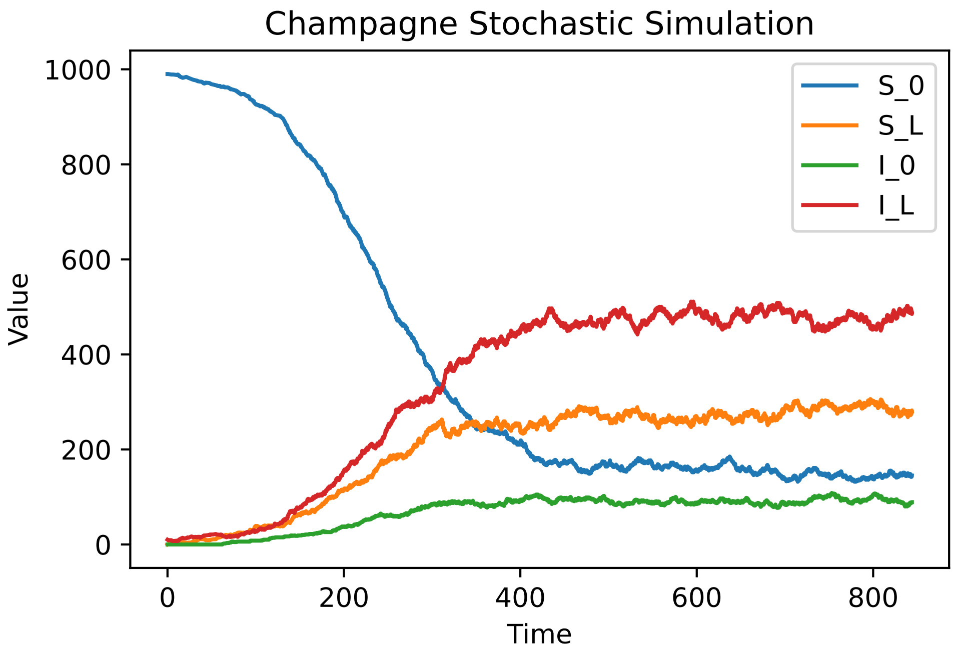
```
        + I_L_to_S_L
        + S_L_to_S_0
        + S_L_to_I_L
    )

    delta_t = np.random.exponential(1 / total_rate)
    new_stages_prob = [
        S_0_to_I_L / total_rate,
        S_0_to_S_L / total_rate,
        I_0_to_S_0 / total_rate,
        I_0_to_I_L / total_rate,
        I_L_to_I_0 / total_rate,
        I_L_to_S_L / total_rate,
        S_L_to_S_0 / total_rate,
        S_L_to_I_L / total_rate,
    ]
    t += delta_t
    silent_incidences = np.random.poisson(
        delta_t * alpha_ * beta_ * lambda_ * (I_L + I_0) * S_0 / N
    )

    new_stages = np.random.choice(
        [
            {
                "t": t,
                "S_0": S_0 - 1,
                "S_L": S_L,
                "I_0": I_0,
                "I_L": I_L + 1,
                "inc_counter": inc_counter + silent_incidences + 1,
            },
            {
                "t": t,
                "S_0": S_0 - 1,
                "S_L": S_L + 1,
                "I_0": I_0,
                "I_L": I_L,
                "inc_counter": inc_counter + silent_incidences + 1,
            },
            {
                "t": t,
                "S_0": S_0 + 1,
```

```
                "S_L": S_L,
                "I_0": I_0 - 1,
                "I_L": I_L,
                "inc_counter": inc_counter + silent_incidences,
            },
            {
                "t": t,
                "S_0": S_0,
                "S_L": S_L,
                "I_0": I_0 - 1,
                "I_L": I_L + 1,
                "inc_counter": inc_counter + silent_incidences,
            },
            {
                "t": t,
                "S_0": S_0,
                "S_L": S_L,
                "I_0": I_0 + 1,
                "I_L": I_L - 1,
                "inc_counter": inc_counter + silent_incidences,
            },
            {
                "t": t,
                "S_0": S_0,
                "S_L": S_L + 1,
                "I_0": I_0,
                "I_L": I_L - 1,
                "inc_counter": inc_counter + silent_incidences,
            },
            {
                "t": t,
                "S_0": S_0 + 1,
                "S_L": S_L - 1,
                "I_0": I_0,
                "I_L": I_L,
                "inc_counter": inc_counter
                + silent_incidences
                + np.random.binomial(1, prop_new),
            },
            {
                "t": t,
                "S_0": S_0,
```

```
                    "S_L": S_L - 1,
                    "I_0": I_0,
                    "I_L": I_L + 1,
                    "inc_counter": inc_counter + silent_incidences + 1,
                },
            ],
            p=new_stages_prob,
        )

        list_of_outcomes.append(new_stages)

        S_0 = new_stages["S_0"]
        I_0 = new_stages["I_0"]
        I_L = new_stages["I_L"]
        S_L = new_stages["S_L"]
        inc_counter = new_stages["inc_counter"]

    outcome_df = pd.DataFrame(list_of_outcomes)
    return outcome_df


champ_samp = champagne_stochastic(
    pv_champ_alpha,
    pv_champ_beta,
    pv_champ_gamma_L,
    pv_champ_lambda,
    pv_champ_f,
    pv_champ_r,
)  # .melt(id_vars='t')
```

Plotting outcome

```
champ_samp.drop("inc_counter", axis=1).plot(x="t", legend=True)
plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Champagne Stochastic Simulation")
plt.savefig("champagne_GP_images/champagne_simulation.pdf")
plt.show()
```



Function that Outputs Final Prevalence

```
def incidence(df, start, days):
    start_ind = df[df["t"].le(start)].index[-1]
    end_ind = df[df["t"].le(start + days)].index[-1]
    incidence_week = df.iloc[end_ind]["inc_counter"] - df.iloc[start_ind]["inc_counter"]
    return incidence_week


def champ_sum_stats(alpha_, beta_, gamma_L, lambda_, f, r):
    champ_df_ = champagne_stochastic(alpha_, beta_, gamma_L, lambda_, f, r)
    fin_t = champ_df_.iloc[-1]["t"]
    first_month_inc = incidence(champ_df_, 0, 30)
    fin_t = champ_df_.iloc[-1]["t"]
    fin_week_inc = incidence(champ_df_, fin_t - 7, 7)
    fin_prev = champ_df_.iloc[-1]["I_0"] + champ_df_.iloc[-1]["I_L"]

    return np.array([fin_prev, first_month_inc, fin_week_inc])


observed_sum_stats = champ_sum_stats(
```

```
    pv_champ_alpha,
    pv_champ_beta,
    pv_champ_gamma_L,
    pv_champ_lambda,
    pv_champ_f,
    pv_champ_r,
)


def discrepency_fn(alpha_, beta_, gamma_L, lambda_, f, r): # best is L1 norm
    x = champ_sum_stats(alpha_, beta_, gamma_L, lambda_, f, r)
    return np.sum(np.abs((x - observed_sum_stats) / observed_sum_stats))
```

Testing the variances across different values of params etc.

```
# samples = 30
# cor_sums = np.zeros(samples)
# for i in range(samples):
#     cor_sums[i] = discrepency_fn(
#         pv_champ_alpha,
#         pv_champ_beta,
#         pv_champ_gamma_L,
#         pv_champ_lambda,
#         pv_champ_f,
#         pv_champ_r,
#     )

# cor_mean = np.mean(cor_sums)
# cor_s_2 = sum((cor_sums - cor_mean) ** 2) / (samples - 1)
# print(cor_mean, cor_s_2)

# doub_sums = np.zeros(samples)
# for i in range(samples):
#     doub_sums[i] = discrepency_fn(
#         2 * pv_champ_alpha,
#         2 * pv_champ_beta,
#         2 * pv_champ_gamma_L,
#         2 * pv_champ_lambda,
#         2 * pv_champ_f,
#         2 * pv_champ_r,
#     )
```

```
# doub_mean = np.mean(doub_sums)
# doub_s_2 = sum((doub_sums - doub_mean) ** 2) / (samples - 1)
# print(doub_mean, doub_s_2)

# half_sums = np.zeros(samples)
# for i in range(samples):
#     half_sums[i] = discrepency_fn(
#         pv_champ_alpha / 2,
#         pv_champ_beta / 2,
#         pv_champ_gamma_L / 2,
#         pv_champ_lambda / 2,
#         pv_champ_f / 2,
#         pv_champ_r / 2,
#     )

# half_mean = np.mean(half_sums)
# half_s_2 = sum((half_sums - half_mean) ** 2) / (samples - 1)
# print(half_mean, half_s_2)

# rogue_sums = np.zeros(samples)
# for i in range(samples):
#     rogue_sums[i] = discrepency_fn(
#         pv_champ_alpha / 2,
#         pv_champ_beta / 2,
#         pv_champ_gamma_L / 2,
#         pv_champ_lambda / 2,
#         pv_champ_f / 2,
#         pv_champ_r / 2,
#     )

# rogue_mean = np.mean(rogue_sums)
# rogue_s_2 = sum((rogue_sums - rogue_mean) ** 2) / (samples - 1)
# print(rogue_mean, rogue_s_2)

# plt.figure(figsize=(7, 4))
# plt.scatter(
#     np.array([half_mean, cor_mean, doub_mean, rogue_mean]),
#     np.array([half_s_2, cor_s_2, doub_s_2, rogue_s_2]),
# )
# plt.title("variance and mean")
# plt.xlabel("mean")
# plt.ylabel("variance")
```

```
# plt.show()
```

Gaussian Process Regression on Final Prevalence Discrepency

```
my_seed = np.random.default_rng(seed=1795)  # For replicability

num_samples = 30

variables_names = ["alpha", "beta", "gamma_L", "lambda", "f", "r"]

pv_champ_alpha = 0.4  # prop of effective care
pv_champ_beta = 0.4  # prop of radical cure
pv_champ_gamma_L = 1 / 223  # liver stage clearance rate
pv_champ_lambda = 0.04  # transmission rate
pv_champ_f = 1 / 72  # relapse frequency
pv_champ_r = 1 / 60  # blood stage clearance rate

samples = np.concatenate(
    (
        my_seed.uniform(low=0, high=1, size=(num_samples, 1)),  # alpha
        my_seed.uniform(low=0, high=1, size=(num_samples, 1)),  # beta
        my_seed.exponential(scale=pv_champ_gamma_L, size=(num_samples, 1)),  # gamma_L
        my_seed.exponential(scale=pv_champ_lambda, size=(num_samples, 1)),  # lambda
        my_seed.exponential(scale=pv_champ_f, size=(num_samples, 1)),  # f
        my_seed.exponential(scale=pv_champ_r, size=(num_samples, 1)),  # r
    ),
    axis=1,
)

LHC_sampler = qmc.LatinHypercube(d=6, seed=my_seed)
LHC_samples = LHC_sampler.random(n=num_samples)
LHC_samples[:, 2] = -pv_champ_gamma_L * np.log(LHC_samples[:, 2])
LHC_samples[:, 3] = -pv_champ_lambda * np.log(LHC_samples[:, 3])
LHC_samples[:, 4] = -pv_champ_f * np.log(LHC_samples[:, 4])
LHC_samples[:, 5] = -pv_champ_r * np.log(LHC_samples[:, 5])

LHC_samples = np.repeat(LHC_samples, 3, axis = 0)

random_indices_df = pd.DataFrame(samples, columns=variables_names)
```

```
LHC_indices_df = pd.DataFrame(LHC_samples, columns=variables_names)

print(random_indices_df.head())
print(LHC_indices_df.head())
```

```
      alpha      beta   gamma_L    lambda         f         r
0  0.201552  0.081511  0.004695  0.017172  0.007355  0.021370
1  0.332324  0.374497  0.003022  0.020210  0.001350  0.002604
2  0.836050  0.570164  0.002141  0.043572  0.001212  0.008367
3  0.566773  0.347186  0.001925  0.016830  0.000064  0.003145
4  0.880603  0.316884  0.000425  0.012374  0.000358  0.003491
      alpha      beta   gamma_L    lambda         f         r
0  0.066680  0.570582  0.001707  0.002226  0.004358  0.003743
1  0.066680  0.570582  0.001707  0.002226  0.004358  0.003743
2  0.066680  0.570582  0.001707  0.002226  0.004358  0.003743
3  0.132042  0.551592  0.013131  0.036829  0.002851  0.002075
4  0.132042  0.551592  0.013131  0.036829  0.002851  0.002075
```

Generate Discrepencies

```
random_discrepencies = LHC_indices_df.apply(
    lambda x: discrepency_fn(
        x["alpha"], x["beta"], x["gamma_L"], x["lambda"], x["f"], x["r"]
    ),
    axis=1,
)

print(random_discrepencies.head())
```

```
0    1.720391
1    1.873389
2    1.916142
3    1.904911
4    1.950292
dtype: float64
```

Differing Methods to Iterate Function

```
# import timeit

# def function1():
#     np.vectorize(champ_sum_stats)(random_indices_df['alpha'],
#     random_indices_df['beta'], random_indices_df['gamma_L'],
#     random_indices_df['lambda'], random_indices_df['f'], random_indices_df['r'])
#     pass

# def function2():
#     random_indices_df.apply(
#         lambda x: champ_sum_stats(
#             x['alpha'], x['beta'], x['gamma_L'], x['lambda'], x['f'], x['r']),
#             axis = 1)
#     pass

# # Time function1
# time_taken_function1 = timeit.timeit(
#     "function1()", globals=globals(), number=100)

# # Time function2
# time_taken_function2 = timeit.timeit(
#     "function2()", globals=globals(), number=100)

# print("Time taken for function1:", time_taken_function1)
# print("Time taken for function2:", time_taken_function2)
```

Time taken for function1: 187.48960775700016 Time taken for function2: 204.06618941299985

Constrain Variables to be Positive

```
constrain_positive = tfb.Shift(np.finfo(np.float64).tiny)(tfb.Exp())
```

```
2024-04-29 09:01:04.012616: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:9
2024-04-29 09:01:04.050501: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot d
Skipping registering GPU devices...
```

Custom Quadratic Mean Function

```
class quad_mean_fn(tf.Module):
    def __init__(self):
        super(quad_mean_fn, self).__init__()
        # self.amp_alpha_mean = tfp.util.TransformedVariable(
        #     bijector=constrain_positive,
        #     initial_value=1.0,
        #     dtype=np.float64,
        #     name="amp_alpha_mean",
        # )
        # self.alpha_tp = tf.Variable(pv_champ_alpha, dtype=np.float64, name="alpha_tp")
        # self.amp_beta_mean = tfp.util.TransformedVariable(
        #     bijector=constrain_positive,
        #     initial_value=0.5,
        #     dtype=np.float64,
        #     name="amp_beta_mean",
        # )
        # self.beta_tp = tf.Variable(pv_champ_beta, dtype=np.float64, name="beta_tp")
        self.amp_gamma_L_mean = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_gamma_L_mean",
        )
        self.gamma_L_tp = tf.Variable(
            pv_champ_gamma_L, dtype=np.float64, name="gamma_L_tp"
        )
        self.amp_lambda_mean = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_lambda_mean",
        )
        self.lambda_tp = tf.Variable(
            pv_champ_lambda, dtype=np.float64, name="lambda_tp"
        )
        self.amp_f_mean = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_f_mean",
```

```
        )
        self.f_tp = tf.Variable(pv_champ_f, dtype=np.float64, name="f_tp")
        self.amp_r_mean = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_r_mean",
        )
        self.r_tp = tf.Variable(pv_champ_r, dtype=np.float64, name="r_tp")
        self.bias_mean = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=50.0,
            dtype=np.float64,
            name="bias_mean",
        )
        # self.bias_mean = tf.Variable(0.0, dtype=np.float64, name="bias_mean")

    def __call__(self, x):
        return (
            self.bias_mean
            # + self.amp_alpha_mean * (x[..., 0] - self.alpha_tp) ** 2
            # + self.amp_beta_mean * (x[..., 1] - self.beta_tp) ** 2
            + self.amp_gamma_L_mean * (x[..., 2] - self.gamma_L_tp) ** 2
            + self.amp_lambda_mean * (x[..., 3] - self.lambda_tp) ** 2
            + self.amp_f_mean * (x[..., 4] - self.f_tp) ** 2
            + self.amp_r_mean * (x[..., 5] - self.r_tp) ** 2
        )
```

Custom Linear Mean Function

```
class lin_mean_fn(tf.Module):
    def __init__(self):
        super(lin_mean_fn, self).__init__()
        # self.amp_alpha_lin = tfp.util.TransformedVariable(
        #     bijector=constrain_positive,
        #     initial_value=1.0,
        #     dtype=np.float64,
        #     name="amp_alpha_lin",
        # )
        # self.amp_beta_lin = tfp.util.TransformedVariable(
```

```
        #     bijector=constrain_positive,
        #     initial_value=0.5,
        #     dtype=np.float64,
        #     name="amp_beta_lin",
        # )
        self.amp_gamma_L_lin = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_gamma_L_lin",
        )
        self.amp_lambda_lin = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_lambda_lin",
        )
        self.amp_f_lin = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_f_lin",
        )
        self.amp_r_lin = tfp.util.TransformedVariable(
            bijector=constrain_positive,
            initial_value=1.0,
            dtype=np.float64,
            name="amp_r_lin",
        )
        # self.bias_lin = tfp.util.TransformedVariable(
        #     bijector=constrain_positive,
        #     initial_value=1.0,
        #     dtype=np.float64,
        #     name="bias_lin",
        # )
        self.bias_lin = tf.Variable(0.0, dtype=np.float64, name="bias_mean")

    def __call__(self, x):
        return (
            self.bias_lin
            # + self.amp_alpha_lin * (x[..., 0])
            # + self.amp_beta_lin * (x[..., 1])
```

```
            + self.amp_gamma_L_lin * (x[..., 2])
            + self.amp_lambda_lin * (x[..., 3])
            + self.amp_f_lin * (x[..., 4])
            + self.amp_r_lin * (x[..., 5])
        )
```

Making the ARD Kernel

```
index_vals = LHC_indices_df.values
obs_vals = random_discrepencies.values

amplitude_champ = tfp.util.TransformedVariable(
    bijector=constrain_positive,
    initial_value=1.0,
    dtype=np.float64,
    name="amplitude_champ",
)

observation_noise_variance_champ = tfp.util.TransformedVariable(
    bijector=constrain_positive,
    initial_value=2,
    dtype=np.float64,
    name="observation_noise_variance_champ",
)
```

```
length_scales_champ = tfp.util.TransformedVariable(
    bijector=tfb.Sigmoid(),
    initial_value=[0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
    dtype=np.float64,
    name="length_scales_champ",
)
```

```
kernel_champ = tfk.FeatureScaled(
    tfk.MaternFiveHalves(amplitude=amplitude_champ),
    scale_diag=length_scales_champ,
)
```

Define the Gaussian Process with Quadratic Mean Function and ARD Kernel

```
# Define Gaussian Process with the custom kernel
champ_GP = tfd.GaussianProcess(
    kernel=kernel_champ,
    observation_noise_variance=observation_noise_variance_champ,
    index_points=index_vals,
    mean_fn=quad_mean_fn(),
)

print(champ_GP.trainable_variables)

Adam_optim = tf.optimizers.Adam(learning_rate=0.01)
```

```
(<tf.Variable 'amplitude_champ:0' shape=() dtype=float64, numpy=0.0>, <tf.Variable 'length_sc
```

Train the Hyperparameters

```
# predictive log stuff
@tf.function(autograph=False, jit_compile=False)
def optimize():
    with tf.GradientTape() as tape:
        K = (
            champ_GP.kernel.matrix(index_vals, index_vals)
            + tf.eye(index_vals.shape[0], dtype=np.float64)
            * observation_noise_variance_champ
        )
        means = champ_GP.mean_fn(index_vals)
        K_inv = tf.linalg.inv(K)
        K_inv_y = K_inv @ tf.reshape(obs_vals - means, shape=[obs_vals.shape[0], 1])
        K_inv_diag = tf.linalg.diag_part(K_inv)
        log_var = tf.math.log(K_inv_diag)
        log_mu = tf.reshape(K_inv_y, shape=[-1]) ** 2
        loss = -tf.math.reduce_sum(log_var - log_mu)
    grads = tape.gradient(loss, champ_GP.trainable_variables)
    Adam_optim.apply_gradients(zip(grads, champ_GP.trainable_variables))
    return loss
```

```
num_iters = 10000

lls_ = np.zeros(num_iters, np.float64)
tolerance = 1e-6  # Set your desired tolerance level
previous_loss = float("inf")

for i in range(num_iters):
    loss = optimize()
    lls_[i] = loss

    # Check if change in loss is less than tolerance
    if abs(loss - previous_loss) < tolerance:
        print(f"Hyperparameter convergence reached at iteration {i+1}.")
        lls_ = lls_[range(i + 1)]
        break

    previous_loss = loss
```

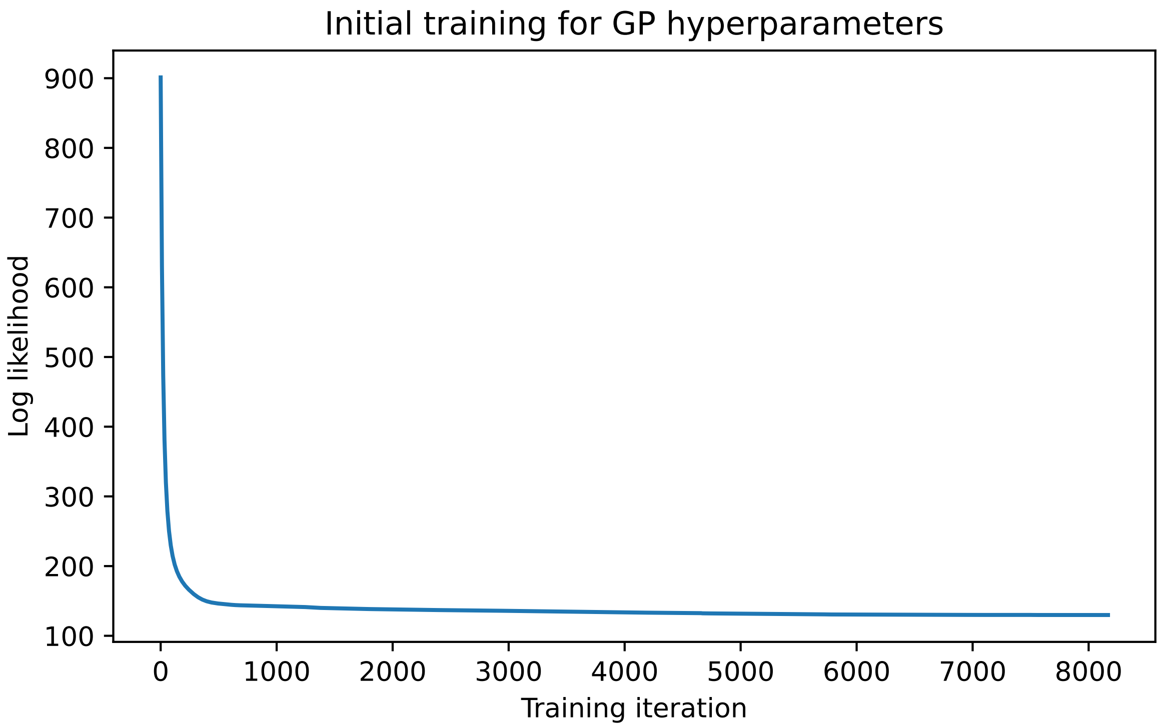
```
Hyperparameter convergence reached at iteration 8168.
```

```
print("Trained parameters:")
for var in champ_GP.trainable_variables:
    if "length" in var.name:
        print(
            "{} is {}\n".format(
                var.name, tfb.Sigmoid().forward(var).numpy().round(3)
            )
        )
    else:
        if "tp" in var.name: # or "bias" in var.name:
            print("{} is {}\n".format(var.name, var.numpy().round(3)))
        else:
            print(
                "{} is {}\n".format(
                    var.name, constrain_positive.forward(var).numpy().round(3)
                )
            )
```

```
Trained parameters:
amplitude_champ:0 is 0.043
```

```
length_scales_champ:0 is [0.989 0.989 0.941 0.5   0.986 0.993]

observation_noise_variance_champ:0 is 2.565

amp_f_mean:0 is 638.455

amp_gamma_L_mean:0 is 966.216

amp_lambda_mean:0 is 996.855

amp_r_mean:0 is 0.011

bias_mean:0 is 0.276

f_tp:0 is 0.007

gamma_L_tp:0 is -0.023

lambda_tp:0 is 0.044

r_tp:0 is 0.009
```

```
plt.figure(figsize=(7, 4))
plt.plot(lls_)
plt.title("Initial training for GP hyperparameters")
plt.xlabel("Training iteration")
plt.ylabel("Log likelihood")
plt.savefig("champagne_GP_images/hyperparam_loss_no_log.pdf")
plt.show()
```



Creating slices across one variable dimension

```
plot_samp_no = 21
plot_gp_no = 200
gp_samp_no = 50
```

```
slice_samples_dict = {
    "alpha_slice_samples": np.repeat(np.concatenate(
        (
            np.linspace(0, 1, plot_samp_no, dtype=np.float64).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_samp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_samp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_samp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_samp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 3, axis = 0),
    "alpha_gp_samples": np.concatenate(
```

```
        (
            np.linspace(0, 1, plot_gp_no, dtype=np.float64).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_gp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_gp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ),
    "beta_slice_samples": np.repeat(np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_samp_no).reshape(-1, 1),  # alpha
            np.linspace(0, 1, plot_samp_no, dtype=np.float64).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_samp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_samp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_samp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 3, axis = 0),
    "beta_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.linspace(0, 1, plot_gp_no, dtype=np.float64).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_gp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_gp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ),
    "gamma_L_slice_samples": np.repeat(np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_samp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_samp_no).reshape(-1, 1),  # beta
            -10*pv_champ_gamma_L
            * np.log(
                np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
            ).reshape(
                -1, 1
            ),  # gamma_L
```

```
            np.repeat(pv_champ_lambda, plot_samp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_samp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 3, axis = 0),
    "gamma_L_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.linspace(
                -10*pv_champ_gamma_L
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[0],
                -10*pv_champ_gamma_L
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[-1], plot_gp_no, dtype=np.float64
            ),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_gp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_gp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ),
    "lambda_slice_samples": np.repeat(np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_samp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_samp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_samp_no).reshape(-1, 1),  # gamma_L
            -pv_champ_lambda
            * np.log(
                np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
            ).reshape(
                -1, 1
            ),  # lambda
            np.repeat(pv_champ_f, plot_samp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 3, axis = 0),
```

```
    "lambda_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.linspace(
                -pv_champ_lambda
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[0],
                -pv_champ_lambda
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[-1], plot_gp_no, dtype=np.float64
            ),  # lambda
            np.repeat(pv_champ_f, plot_gp_no).reshape(-1, 1),  # f
            np.repeat(pv_champ_r, plot_gp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ),
    "f_slice_samples": np.repeat(np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_samp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_samp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_samp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_samp_no).reshape(-1, 1),  # lambda
            -10*pv_champ_f
            * np.log(
                np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
            ).reshape(
                -1, 1
            ),  # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 3, axis = 0),
    "f_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1, 1),  # lambda
```

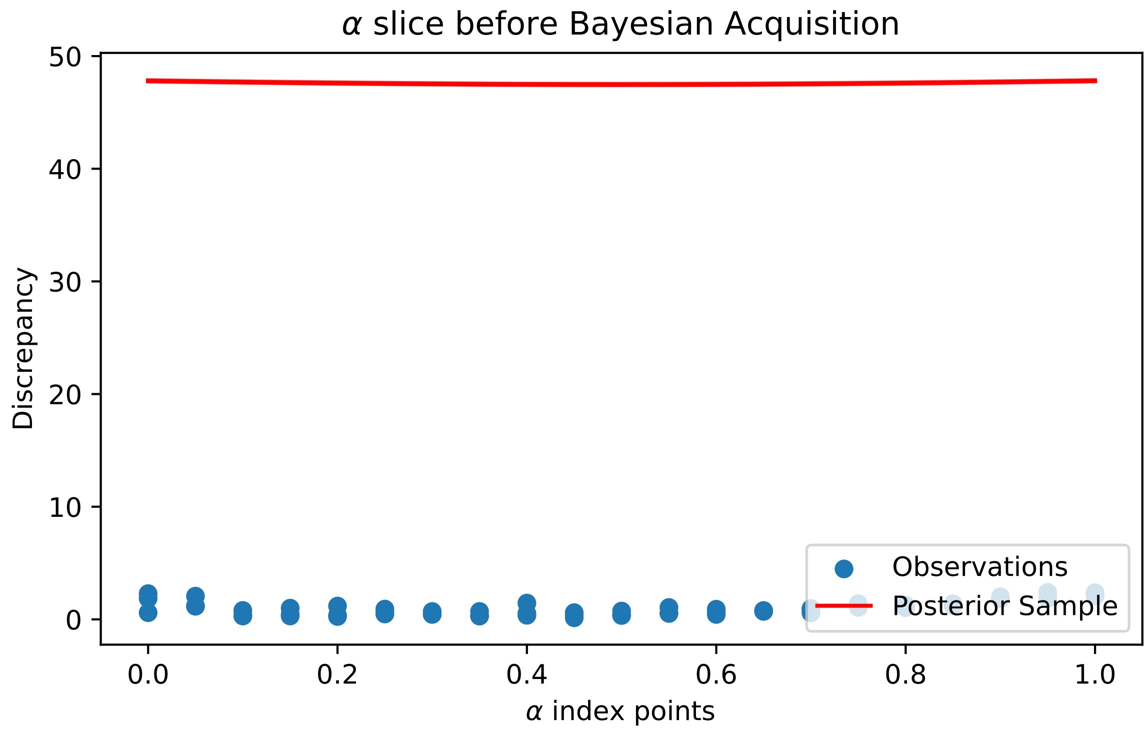
```
            np.linspace(
                -10*pv_champ_f
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[0],
                -10*pv_champ_f
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[-1], plot_gp_no, dtype=np.float64
            ),  # f
            np.repeat(pv_champ_r, plot_gp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ),
    "r_slice_samples": np.repeat(np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_samp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_samp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_samp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_samp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_samp_no).reshape(-1, 1),  # f
            -2*pv_champ_r
            * np.log(
                np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
            ).reshape(
                -1, 1
            ),  # r
        ),
        axis=1,
    ), 3, axis = 0),
    "r_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1, 1),  # lambda
            np.repeat(pv_champ_f, plot_gp_no).reshape(-1, 1),  # f
            np.linspace(
                -2*pv_champ_r
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[0],
```

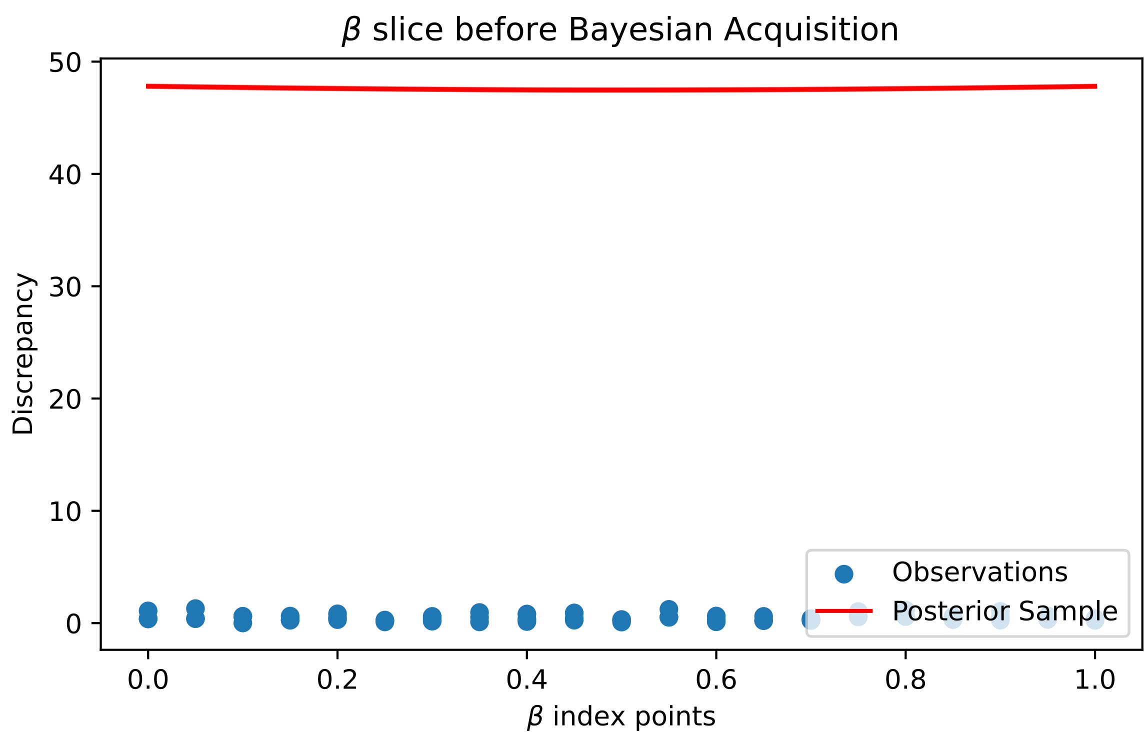
```
                -2*pv_champ_r
                * np.log(
                    np.linspace(0, 1, plot_samp_no + 2, dtype=np.float64)[1:-1]
                ).reshape(-1, 1)[-1], plot_gp_no, dtype=np.float64
            ),  # r
        ),
        axis=1,
    ),
}
```

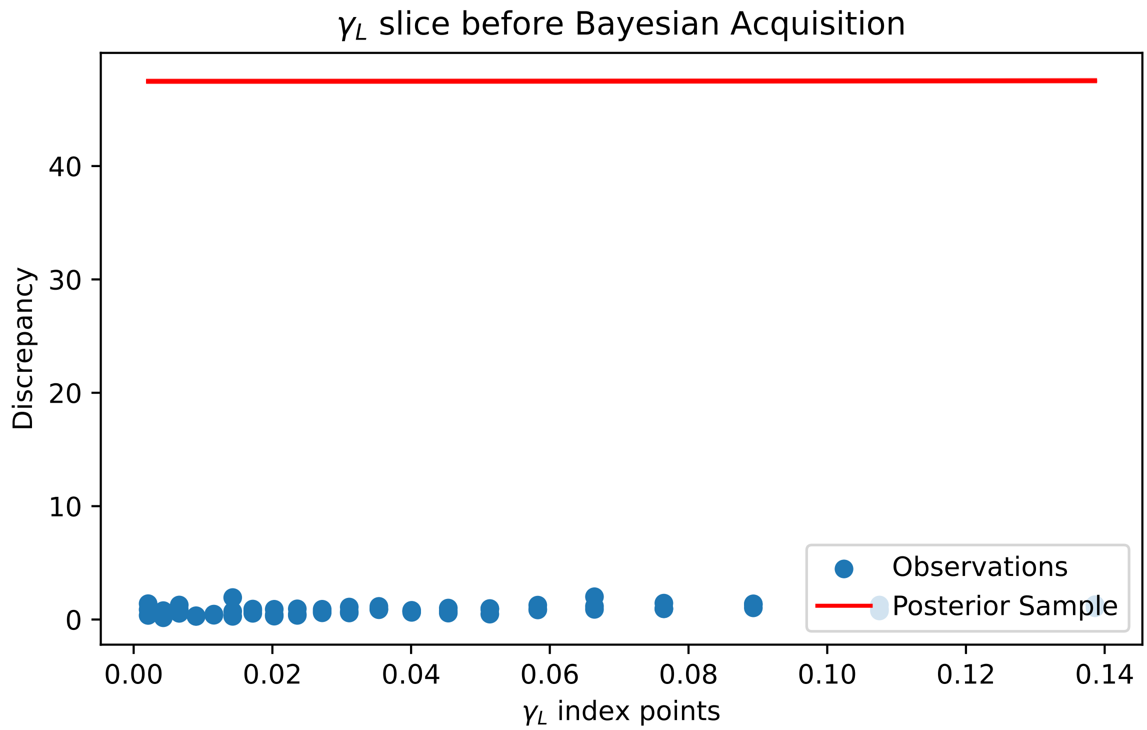
Plotting the GPs across different slices

```
GP_seed = tfp.random.sanitize_seed(4362)
vars = ["alpha", "beta", "gamma_L", "lambda", "f", "r"]
slice_indices_dfs_dict = {}
slice_index_vals_dict = {}
slice_discrepencies_dict = {}

for var in vars:
    val_df = pd.DataFrame(
        slice_samples_dict[var + "_slice_samples"], columns=variables_names
    )
    slice_indices_dfs_dict[var + "_slice_indices_df"] = val_df
    slice_index_vals_dict[var + "_slice_index_vals"] = val_df.values
    discreps = val_df.apply(
        lambda x: discrepency_fn(
            x["alpha"], x["beta"], x["gamma_L"], x["lambda"], x["f"], x["r"]
        ),
        axis=1,
    )
    slice_discrepencies_dict[var + "_slice_discrepencies"] = discreps


    gp_samples_df = pd.DataFrame(
        slice_samples_dict[var + "_gp_samples"], columns=variables_names
    )
    slice_indices_dfs_dict[var + "_gp_indices_df"] = gp_samples_df
    slice_index_vals_dict[var + "_gp_index_vals"] = gp_samples_df.values

    champ_GP_reg = tfd.GaussianProcessRegressionModel(
```

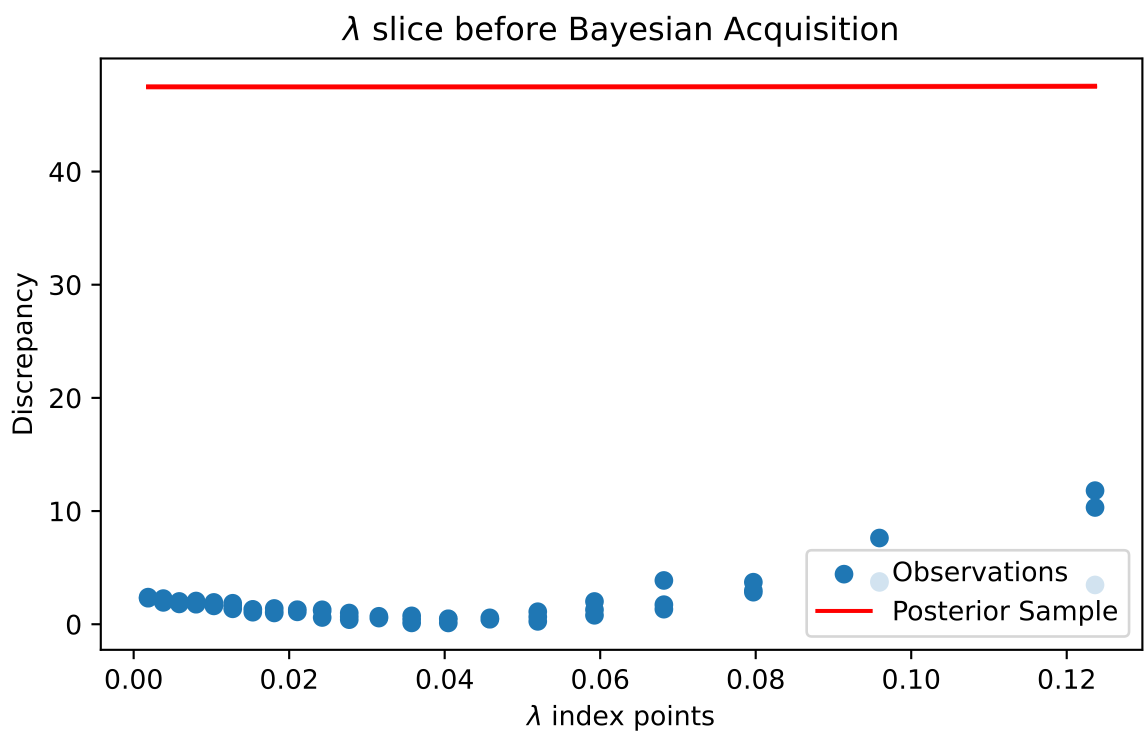
```
        kernel=kernel_champ,
        index_points=gp_samples_df.values,
        observation_index_points=index_vals,
        observations=obs_vals,
        observation_noise_variance=observation_noise_variance_champ,
        predictive_noise_variance=0.0,
        mean_fn=quad_mean_fn(),
    )
    GP_samples = champ_GP_reg.sample(gp_samp_no, seed=GP_seed)

    plt.figure(figsize=(7, 4))
    plt.scatter(
        val_df[var].values,
        discreps,
        label="Observations",
    )
    for i in range(gp_samp_no):
        plt.plot(
            gp_samples_df[var].values,
            GP_samples[i, :],
            c="r",
            alpha=0.1,
            label="Posterior Sample" if i == 0 else None,
        )
    leg = plt.legend(loc="lower right")
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    if var in ["f", "r"]:
        plt.xlabel("$" + var + "$ index points")
        plt.title("$" + var + "$ slice before Bayesian Acquisition")
    else:
        plt.xlabel("$\\" + var + "$ index points")
        plt.title("$\\" + var + "$ slice before Bayesian Acquisition")
    # if var not in ["alpha", "beta"]:
    #     plt.xscale("log", base=np.e)
    plt.ylabel("Discrepancy")
    plt.savefig("champagne_GP_images/initial_" + var + "_slice_no_log.pdf")
    plt.show()
```


α slice before Bayesian Acquisition
Discrepancy
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample

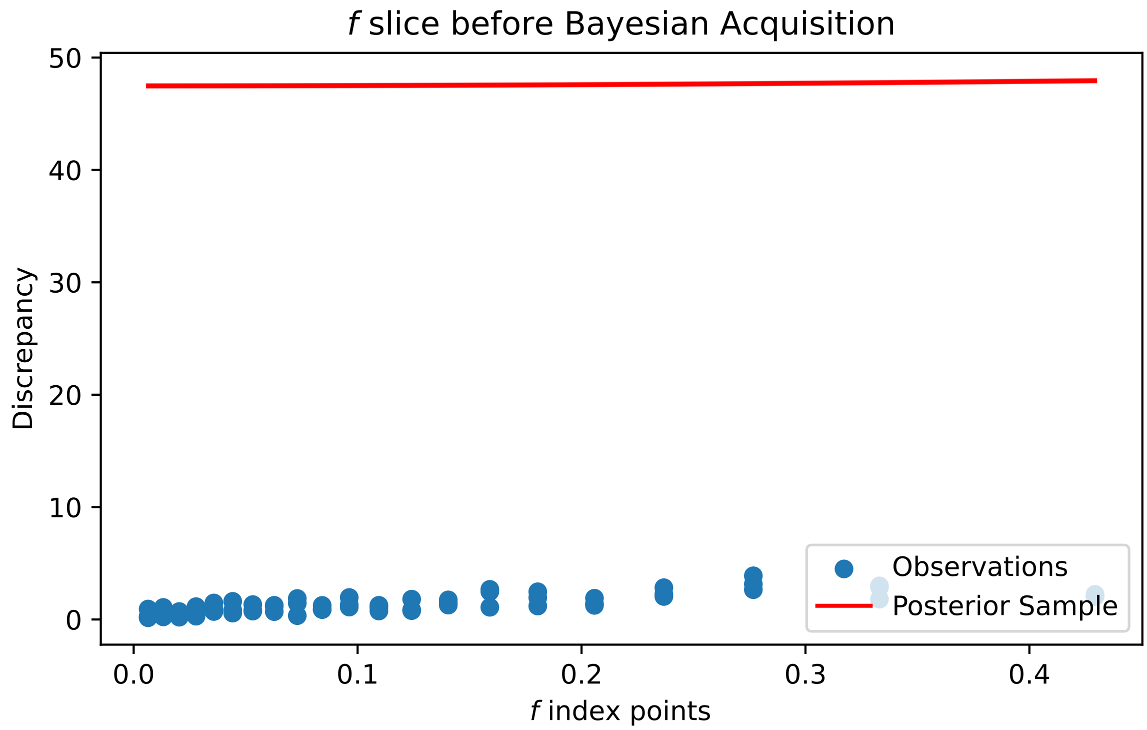

β slice before Bayesian Acquisition
Discrepancy
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

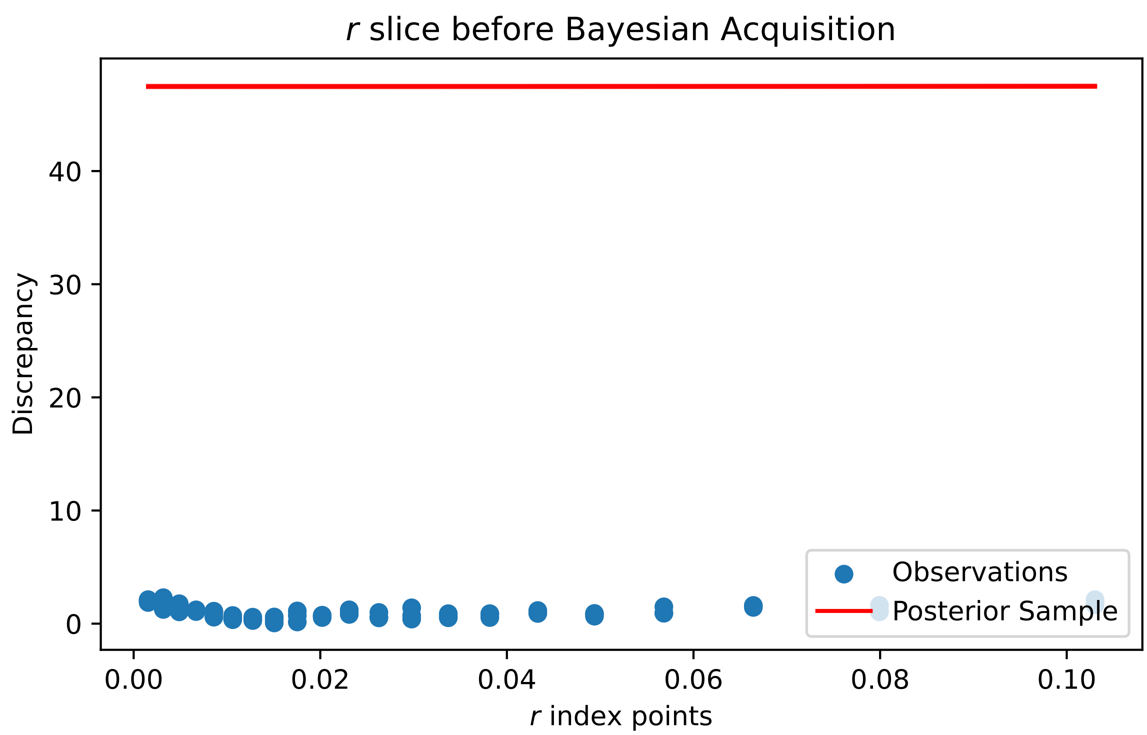


γ_L slice before Bayesian Acquisition
Discrepancy
0
10
20
30
40
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γ_L index points
Observations
Posterior Sample



λ slice before Bayesian Acquisition
Discrepancy
0
10
20
30
40
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample


f slice before Bayesian Acquisition
Discrepancy
f index points
Observations
Posterior Sample


r slice before Bayesian Acquisition
Discrepancy
r index points
Observations
Posterior Sample

Acquiring the next datapoint to test

Proof that .variance returns what we need in acquisition function

```
new_guess = np.array([0.4, 0.4, 0.004, 0.04, 0.01, 0.17])
mean_t = champ_GP_reg.mean_fn(new_guess)
variance_t = champ_GP_reg.variance(index_points=[new_guess])

kernel_self = kernel_champ.apply(new_guess, new_guess)
kernel_others = kernel_champ.apply(new_guess, index_vals)
K = kernel_champ.matrix(
    index_vals, index_vals
) + observation_noise_variance_champ * np.identity(index_vals.shape[0])
inv_K = np.linalg.inv(K)
print("Self Kernel is {}".format(kernel_self.numpy().round(3)))
print("Others Kernel is {}".format(kernel_others.numpy().round(3)))
print(inv_K)
my_var_t = kernel_self - kernel_others.numpy() @ inv_K @ kernel_others.numpy()

print("Variance function is {}".format(variance_t.numpy().round(3)))
print("Variance function is {}".format(my_var_t.numpy().round(3)))
```

```
Self Kernel is 0.002
Others Kernel is [0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.001 0.001 0.001
 0.001 0.001 0.001 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002
 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002
 0.001 0.001 0.001 0.001 0.001 0.001 0.002 0.002 0.002 0.001 0.001 0.001
 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.001 0.001 0.001
 0.001 0.001 0.001 0.001 0.001 0.001 0.002 0.002 0.002 0.002 0.002 0.002
 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002 0.002
 0.002 0.002 0.002 0.002 0.002 0.002]
[[ 3.89623140e-01 -2.74984276e-04 -2.74984276e-04 ... -1.83930098e-04
  -1.83930098e-04 -1.83930098e-04]
 [-2.74984276e-04  3.89623140e-01 -2.74984276e-04 ... -1.83930098e-04
  -1.83930098e-04 -1.83930098e-04]
 [-2.74984276e-04 -2.74984276e-04  3.89623140e-01 ... -1.83930098e-04
  -1.83930098e-04 -1.83930098e-04]
 ...
 [-1.83930098e-04 -1.83930098e-04 -1.83930098e-04 ...  3.89624416e-01
  -2.73708258e-04 -2.73708258e-04]
 [-1.83930098e-04 -1.83930098e-04 -1.83930098e-04 ... -2.73708258e-04
```

```
   3.89624416e-01 -2.73708258e-04]
 [-1.83930098e-04 -1.83930098e-04 -1.83930098e-04 ... -2.73708258e-04
  -2.73708258e-04  3.89624416e-01]]
Variance function is [0.002]
Variance function is 0.002
```

Loss function

```
next_alpha = tfp.util.TransformedVariable(
    initial_value=0.5,
    bijector=tfb.Sigmoid(),
    dtype=np.float64,
    name="next_alpha",
)

next_beta = tfp.util.TransformedVariable(
    initial_value=0.5,
    bijector=tfb.Sigmoid(),
    dtype=np.float64,
    name="next_beta",
)

next_gamma_L = tfp.util.TransformedVariable(
    initial_value=0.1,
    bijector=constrain_positive,
    dtype=np.float64,
    name="next_gamma_L",
)

next_lambda = tfp.util.TransformedVariable(
    initial_value=0.1,
    bijector=constrain_positive,
    dtype=np.float64,
    name="next_lambda",
)

next_f = tfp.util.TransformedVariable(
    initial_value=0.1,
    bijector=constrain_positive,
    dtype=np.float64,
    name="next_f",
```

```
)

next_r = tfp.util.TransformedVariable(
    initial_value=0.1,
    bijector=constrain_positive,
    dtype=np.float64,
    name="next_r",
)

next_vars = [
    v.trainable_variables[0]
    for v in [next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]
]
```

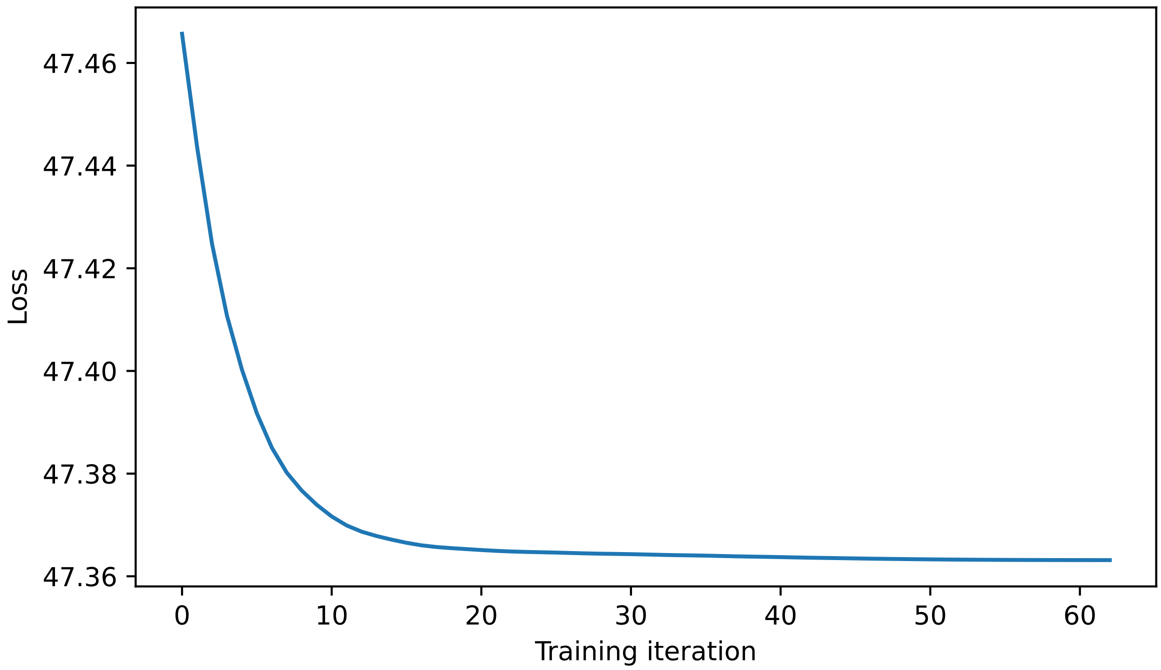
```
Adam_optim = tf.optimizers.Adam(learning_rate=0.1)


@tf.function(autograph=False, jit_compile=False)
def optimize():
    with tf.GradientTape() as tape:
        next_guess = tf.reshape(
            [
                tfb.Sigmoid().forward(next_vars[0]),
                tfb.Sigmoid().forward(next_vars[1]),
                constrain_positive.forward(next_vars[2]),
                constrain_positive.forward(next_vars[3]),
                constrain_positive.forward(next_vars[4]),
                constrain_positive.forward(next_vars[5]),
            ],
            [1, 6],
        )
        mean_t = champ_GP_reg.mean_fn(next_guess)
        std_t = champ_GP_reg.stddev(index_points=next_guess)
        loss = tf.squeeze(mean_t - 1.7 * std_t)
    grads = tape.gradient(loss, next_vars)
    Adam_optim.apply_gradients(zip(grads, next_vars))
    return loss


num_iters = 10000

lls_ = np.zeros(num_iters, np.float64)
```

```python
tolerance = 1e-6  # Set your desired tolerance level
previous_loss = float("inf")

for i in range(num_iters):
    loss = optimize()
    lls_[i] = loss

    # Check if change in loss is less than tolerance
    if abs(loss - previous_loss) < tolerance:
        print(f"Acquisition function convergence reached at iteration {i+1}.")
        lls_ = lls_[range(i + 1)]
        break

    previous_loss = loss

print("Trained parameters:")
for var in next_vars:
    if ("alpha" in var.name) | ("beta" in var.name):
        print(
            "{} is {}".format(var.name, (tfb.Sigmoid().forward(var).numpy().round(3)))
        )
    else:
        print(
            "{} is {}".format(
                var.name, constrain_positive.forward(var).numpy().round(3)
            )
        )
```

```
Acquisition function convergence reached at iteration 63.
Trained parameters:
next_alpha:0 is 0.502
next_beta:0 is 0.508
next_gamma_L:0 is 0.012
next_lambda:0 is 0.037
next_f:0 is 0.014
next_r:0 is 0.016
```

```python
plt.figure(figsize=(7, 4))
plt.plot(lls_)
plt.xlabel("Training iteration")
plt.ylabel("Loss")
```

```python
plt.savefig("champagne_GP_images/bolfi_optim_loss_no_log.pdf")
plt.show()
```




```python
def update_GP():
    @tf.function
    def opt_GP():
        with tf.GradientTape() as tape:
            K = (
                champ_GP.kernel.matrix(index_vals, index_vals)
                + tf.eye(index_vals.shape[0], dtype=np.float64)
                * observation_noise_variance_champ
            )
            means = champ_GP.mean_fn(index_vals)
            K_inv = tf.linalg.inv(K)
            K_inv_y = K_inv @ tf.reshape(obs_vals - means, shape=[obs_vals.shape[0], 1])
            K_inv_diag = tf.linalg.diag_part(K_inv)
            log_var = tf.math.log(K_inv_diag)
            log_mu = tf.reshape(K_inv_y, shape=[-1]) ** 2
            loss = -tf.math.reduce_sum(log_var - log_mu)
        grads = tape.gradient(loss, champ_GP.trainable_variables)
        optimizer_slow.apply_gradients(zip(grads, champ_GP.trainable_variables))
```

```
        return loss

    num_iters = 10000

    lls_ = np.zeros(num_iters, np.float64)
    tolerance = 1e-6  # Set your desired tolerance level
    previous_loss = float("inf")

    for i in range(num_iters):
        loss = opt_GP()
        lls_[i] = loss.numpy()

        # Check if change in loss is less than tolerance
        if abs(loss - previous_loss) < tolerance:
            print(f"Hyperparameter convergence reached at iteration {i+1}.")
            lls_ = lls_[range(i + 1)]
            break

        previous_loss = loss
    for var in optimizer_slow.variables:
        var.assign(tf.zeros_like(var))


def update_var_UCB():
    @tf.function
    def opt_var():
        with tf.GradientTape() as tape:
            next_guess = tf.reshape(
                [
                    tfb.Sigmoid().forward(next_vars[0]),
                    tfb.Sigmoid().forward(next_vars[1]),
                    constrain_positive.forward(next_vars[2]),
                    constrain_positive.forward(next_vars[3]),
                    constrain_positive.forward(next_vars[4]),
                    constrain_positive.forward(next_vars[5]),
                ],
                [1, 6],
            )
            mean_t = champ_GP_reg.mean_fn(next_guess)
            std_t = champ_GP_reg.stddev(index_points=next_guess)
            loss = tf.squeeze(mean_t - eta_t * std_t)
        grads = tape.gradient(loss, next_vars)
```

```
        optimizer_fast.apply_gradients(zip(grads, next_vars))
        return loss

    num_iters = 10000

    lls_ = np.zeros(num_iters, np.float64)
    tolerance = 1e-6  # Set your desired tolerance level
    previous_loss = float("inf")

    for i in range(num_iters):
        loss = opt_var()
        lls_[i] = loss

        # Check if change in loss is less than tolerance
        if abs(loss - previous_loss) < tolerance:
            print(f"Acquisition function convergence reached at iteration {i+1}.")
            lls_ = lls_[range(i + 1)]
            break

        previous_loss = loss
    print(loss)
    for var in optimizer_fast.variables:
        var.assign(tf.zeros_like(var))


def update_var_EI():
    @tf.function
    def opt_var():
        with tf.GradientTape() as tape:
            next_guess = tf.reshape(
                [
                    tfb.Sigmoid().forward(next_vars[0]),
                    tfb.Sigmoid().forward(next_vars[1]),
                    constrain_positive.forward(next_vars[2]),
                    constrain_positive.forward(next_vars[3]),
                    constrain_positive.forward(next_vars[4]),
                    constrain_positive.forward(next_vars[5]),
                ],
                [1, 6],
            )
            mean_t = champ_GP_reg.mean_fn(next_guess)
            std_t = champ_GP_reg.stddev(index_points=next_guess)
```

```
            delt = min_obs - mean_t
            loss = -delt * tfd.Normal(0, std_t).cdf(delt) - std_t * champ_GP_reg.prob(
                delt, index_points=next_guess
            )
        grads = tape.gradient(loss, next_vars)
        optimizer_fast.apply_gradients(zip(grads, next_vars))
        return loss

    num_iters = 10000

    lls_ = np.zeros(num_iters, np.float64)
    tolerance = 1e-6  # Set your desired tolerance level
    previous_loss = float("inf")

    for i in range(num_iters):
        loss = opt_var()
        lls_[i] = loss

        # Check if change in loss is less than tolerance
        if abs(loss - previous_loss) < tolerance:
            print(f"Acquisition function convergence reached at iteration {i+1}.")
            lls_ = lls_[range(i + 1)]
            break

        previous_loss = loss
    print(loss)
    for var in optimizer_fast.variables:
        var.assign(tf.zeros_like(var))


# EI = tfp_acq.GaussianProcessExpectedImprovement(champ_GP_reg, obs_vals)


def new_eta_t(t, d, exploration_rate):
    return np.log((t + 1) ** (d / 2 + 2) * np.pi**2 / (3 * exploration_rate))
```

```
# exploration_rate = 0.00000001
d = 6
update_freq = 20  # how many iterations before updating GP hyperparams
# eta_t = tf.Variable(0,dtype=np.float64, name = "eta_t")

for t in range(201):
```

```
    min_obs = min(champ_GP_reg.mean_fn(index_vals))
    optimizer_fast = tf.optimizers.Adam(learning_rate=0.01)
    optimizer_slow = tf.optimizers.Adam()
    # eta_t.assign(new_eta_t(t, d, exploration_rate))
    print("Iteration " + str(t))
    # print(eta_t)

    for var in next_vars:
        var.assign(my_seed.uniform(0, 1))

    # update_var_UCB()
    update_var_EI()

    new_params = np.array(
        [
            next_alpha.numpy(),
            next_beta.numpy(),
            next_gamma_L.numpy(),
            next_lambda.numpy(),
            next_f.numpy(),
            next_r.numpy(),
        ]
    ).reshape(1, -1)
    print(new_params)

    for repeats in range(3):
        new_discrepency = discrepency_fn(
            next_alpha.numpy(),
            next_beta.numpy(),
            next_gamma_L.numpy(),
            next_lambda.numpy(),
            next_f.numpy(),
            next_r.numpy(),
        )

        index_vals = np.append(
            index_vals,
            new_params,
            axis=0,
        )
        obs_vals = np.append(obs_vals, new_discrepency)
```

```
    if t % update_freq == 0:
        champ_GP = tfd.GaussianProcess(
            kernel=kernel_champ,
            observation_noise_variance=observation_noise_variance_champ,
            index_points=index_vals,
            mean_fn=quad_mean_fn(),
        )
        update_GP()

    champ_GP_reg = tfd.GaussianProcessRegressionModel(
        kernel=kernel_champ,
        observation_index_points=index_vals,
        observations=obs_vals,
        observation_noise_variance=observation_noise_variance_champ,
        predictive_noise_variance=0.0,
        mean_fn=quad_mean_fn(),
    )

    if (t > 0) & (t % 50 == 0):
        print("Trained parameters:")
        for train_var in champ_GP.trainable_variables:
            if "length" in train_var.name:
                print(
                    "{} is {}\n".format(
                        train_var.name,
                        tfb.Sigmoid().forward(train_var).numpy().round(3),
                    )
                )
            else:
                if "tp" in train_var.name:  # or "bias" in var.name:
                    print(
                        "{} is {}\n".format(train_var.name, train_var.numpy().round(3))
                    )
                else:
                    print(
                        "{} is {}\n".format(
                            train_var.name,
                            constrain_positive.forward(train_var).numpy().round(3),
                        )
                    )
        for var in vars:
            champ_GP_reg = tfd.GaussianProcessRegressionModel(
```

```
        kernel=kernel_champ,
        index_points=slice_indices_dfs_dict[var + "_gp_indices_df"].values,
        observation_index_points=index_vals,
        observations=obs_vals,
        observation_noise_variance=observation_noise_variance_champ,
        predictive_noise_variance=0.0,
        mean_fn=quad_mean_fn(),
    )
    GP_samples = champ_GP_reg.sample(gp_samp_no, seed=GP_seed)

    plt.figure(figsize=(7, 4))
    plt.scatter(
        slice_indices_dfs_dict[var + "_slice_indices_df"][var].values,
        slice_discrepencies_dict[var + "_slice_discrepencies"],
        label="Observations",
    )
    for i in range(gp_samp_no):
        plt.plot(
            slice_indices_dfs_dict[var + "_gp_indices_df"][var].values,
            GP_samples[i, :],
            c="r",
            alpha=0.1,
            label="Posterior Sample" if i == 0 else None,
        )
    leg = plt.legend(loc="lower right")
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    if var in ["f", "r"]:
        plt.xlabel("$" + var + "$ index points")
        plt.title(
            "$" + var + "$ slice after " + str(t) + " Bayesian acquisitions"
        )
    else:
        plt.xlabel("$\\" + var + "$ index points")
        plt.title(
            "$\\" + var + "$ slice after " + str(t) + " Bayesian acquisitions"
        )
    plt.ylabel("Discrepancy")
    plt.savefig(
        "champagne_GP_images/"
        + var
        + "_slice_"
```

```python
                + str(t)
                + "_bolfi_updates_no_log.pdf"
            )
            plt.show()

# print(index_vals[-600,])
# print(index_vals[-400,])
print(index_vals[-200,])
print(index_vals[-80,])
print(index_vals[-40,])
print(index_vals[-20,])
print(index_vals[-8,])
print(index_vals[-4,])
print(index_vals[-2,])
print(index_vals[-1,])
```

```
Iteration 0
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63800288 0.69090193 2.51999204 2.21625293 1.34504516 1.22744681]]
Iteration 1
Acquisition function convergence reached at iteration 2.
tf.Tensor([5.63221263e-49], shape=(1,), dtype=float64)
[[0.69889303 0.64569378 1.25099375 1.9243244  2.09770682 1.50463147]]
Iteration 2
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.55407211e-148], shape=(1,), dtype=float64)
[[0.73102701 0.60156065 2.20603773 2.29881551 1.02495331 1.20781954]]
Iteration 3
Acquisition function convergence reached at iteration 2.
tf.Tensor([6.56171462e-92], shape=(1,), dtype=float64)
[[0.55727758 0.57355242 1.56701219 1.85079406 2.11405086 2.34726524]]
Iteration 4
WARNING:tensorflow:5 out of the last 9 calls to <function update_var_EI.<locals>.opt_var at 0
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.24164497e-110], shape=(1,), dtype=float64)
[[0.58732828 0.68319363 1.78067636 1.21171859 1.95754198 2.40954623]]
Iteration 5
WARNING:tensorflow:6 out of the last 11 calls to <function update_var_EI.<locals>.opt_var at
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.3730797e-66], shape=(1,), dtype=float64)
[[0.52809793 0.67159976 2.0017829  1.06069485 2.64298787 1.16249903]]
```

```
Iteration 6
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.9069463e-113], shape=(1,), dtype=float64)
[[0.69144252 0.55553244 1.68891552 2.42500358 2.02999478 1.3180664 ]]
Iteration 7
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6462828  0.71675221 1.81054819 1.98839347 1.48508162 1.72983985]]
Iteration 8
Acquisition function convergence reached at iteration 2.
tf.Tensor([9.74214672e-76], shape=(1,), dtype=float64)
[[0.5534323  0.50060997 2.37191682 1.09410505 2.15095454 1.56193683]]
Iteration 9
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69740798 0.56003946 2.69511706 2.05259935 1.22267903 1.49289144]]
Iteration 10
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6833894  0.597956   1.55400467 2.37134255 1.0119667  2.26333918]]
Iteration 11
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.50737154e-85], shape=(1,), dtype=float64)
[[0.51838277 0.62680296 1.10192117 1.20472481 1.89069859 2.19166226]]
Iteration 12
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68004424 0.69236939 1.63339882 2.05474525 2.3553287  2.38390074]]
Iteration 13
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56630201 0.61837014 2.6414153  2.6661315  1.08630451 2.01653936]]
Iteration 14
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66809366 0.56941472 1.7371747  1.41905056 1.87133991 1.4589311 ]]
Iteration 15
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63394445 0.52921898 2.2000986  1.2677828  1.92020911 1.42543034]]
Iteration 16
Acquisition function convergence reached at iteration 2.
tf.Tensor([5.85349253e-245], shape=(1,), dtype=float64)
```

```
[[0.65492725 0.68760049 1.4473865  1.18605592 1.62950567 1.63831707]]
Iteration 17
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60990846 0.51133957 1.85303591 1.73771294 1.27609033 1.29757276]]
Iteration 18
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.7048621  0.69487606 1.61111799 2.55019294 1.22759094 1.8283634 ]]
Iteration 19
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58387646 0.72571213 2.29928578 2.10340745 1.34993027 1.36573463]]
Iteration 20
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.7138113  0.59841602 1.51722568 1.12614401 1.44220433 1.24077988]]
Iteration 21
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.89600787e-129], shape=(1,), dtype=float64)
[[0.60701943 0.50046612 2.33723313 1.01869381 2.53763321 2.12680462]]
Iteration 22
Acquisition function convergence reached at iteration 2.
tf.Tensor([4.39270695e-194], shape=(1,), dtype=float64)
[[0.5174011  0.72395477 1.66664208 1.13485907 2.0573769  1.08288966]]
Iteration 23
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72322201 0.50972547 1.61255845 1.48754154 2.0686117  1.71769137]]
Iteration 24
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.49185554e-242], shape=(1,), dtype=float64)
[[0.50490546 0.55193585 2.18686502 1.16351371 1.12537832 1.32537161]]
Iteration 25
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.03977362e-70], shape=(1,), dtype=float64)
[[0.66774921 0.60637776 1.87976192 1.48836116 1.04325125 2.71730405]]
Iteration 26
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50011707 0.6821248  1.70565486 2.19134961 2.06032478 1.20530345]]
Iteration 27
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([3.09052083e-91], shape=(1,), dtype=float64)
[[0.61479899 0.72232711 1.07166355 1.09966979 1.02272351 1.96082034]]
Iteration 28
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6631177  0.5474718  1.41854215 2.27269662 1.61649594 1.42612879]]
Iteration 29
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64599056 0.7260329  1.64334625 1.3323327  2.57227346 1.52398631]]
Iteration 30
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69382197 0.65150091 1.22126102 1.77199048 2.65424017 1.08371202]]
Iteration 31
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63028279 0.57230114 1.22653578 1.91941141 1.26311519 1.59568924]]
Iteration 32
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.97443333e-98], shape=(1,), dtype=float64)
[[0.61660312 0.61086592 1.04284522 1.94605287 1.80548471 2.37655773]]
Iteration 33
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63449945 0.64728445 1.41005527 1.65825955 2.39108106 1.09430866]]
Iteration 34
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69263774 0.66827808 1.80026339 1.72431878 2.61152438 1.93855469]]
Iteration 35
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68442483 0.60503647 1.45026922 1.31066689 2.53555628 1.71387914]]
Iteration 36
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67690011 0.70490733 1.72615039 1.16247076 1.96458256 2.50454244]]
Iteration 37
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.30948966e-136], shape=(1,), dtype=float64)
[[0.62825212 0.68896543 2.37832913 1.50207226 1.82046888 1.19914609]]
Iteration 38
```

```
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59641713 0.70262623 2.35771903 1.62780612 1.01325949 1.28123744]]
Iteration 39
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.70984844 0.55820902 1.69579414 2.16636797 1.01030428 2.53633482]]
Iteration 40
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.12577642e-86], shape=(1,), dtype=float64)
[[0.64637151 0.5623575  1.60258503 1.18706283 2.4496113  2.49852122]]
Iteration 41
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64913495 0.68702442 1.70583503 1.27159821 1.83711157 2.43156253]]
Iteration 42
Acquisition function convergence reached at iteration 2.
tf.Tensor([9.51786998e-268], shape=(1,), dtype=float64)
[[0.67050996 0.5491689  1.05976884 2.54023377 2.30792963 2.30215244]]
Iteration 43
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.50521806e-98], shape=(1,), dtype=float64)
[[0.63600357 0.52720724 2.69027998 2.42834094 2.46614489 1.08257133]]
Iteration 44
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.42533506e-89], shape=(1,), dtype=float64)
[[0.60473154 0.50753608 1.94214961 2.08373465 1.90698214 1.30265072]]
Iteration 45
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.48886957e-297], shape=(1,), dtype=float64)
[[0.59407662 0.55148789 1.27903156 1.1763854  2.12622838 1.0465837 ]]
Iteration 46
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.99705682e-273], shape=(1,), dtype=float64)
[[0.70072331 0.5519926  1.08902838 2.03847586 2.0164681  1.62950519]]
Iteration 47
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66799229 0.66449123 1.63709242 2.38419506 1.04593624 1.37414969]]
Iteration 48
Acquisition function convergence reached at iteration 2.
tf.Tensor([5.80054826e-231], shape=(1,), dtype=float64)
[[0.5821719  0.53803069 1.15532403 2.31197176 2.1716707  2.32610277]]
```

```
Iteration 49
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.11739974e-125], shape=(1,), dtype=float64)
[[0.50995685 0.62913069 1.30265208 1.42971853 1.60443781 2.11466098]]
Iteration 50
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66778884 0.65873192 1.24244069 1.0308924  1.66993292 1.43331981]]
Trained parameters:
amplitude_champ:0 is 17.488

length_scales_champ:0 is [1.    1.    0.059 0.901 0.972 0.636]

observation_noise_variance_champ:0 is 40.472

amp_f_mean:0 is 17.846

amp_gamma_L_mean:0 is 48.669

amp_lambda_mean:0 is 123.929

amp_r_mean:0 is 0.241

bias_mean:0 is 10.011

f_tp:0 is -0.914

gamma_L_tp:0 is 0.71

lambda_tp:0 is -0.353

r_tp:0 is 2.555

Iteration 51
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.35762402e-57], shape=(1,), dtype=float64)
[[0.51572273 0.68600667 1.89507177 1.20365008 2.6030326  2.58197246]]
Iteration 52
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60991356 0.63890544 2.61919401 2.47696179 2.61564323 1.13525731]]
Iteration 53
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([4.45927375e-28], shape=(1,), dtype=float64)
[[0.62404897 0.60323984 1.02125742 1.09454366 1.29661677 1.51825062]]
Iteration 54
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67098954 0.67550133 1.10955863 1.12977415 2.37538148 2.14714873]]
Iteration 55
Acquisition function convergence reached at iteration 2.
tf.Tensor([5.13656131e-249], shape=(1,), dtype=float64)
[[0.57479837 0.65980093 1.51394659 1.93580648 2.44447462 1.53698827]]
Iteration 56
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.14084313e-65], shape=(1,), dtype=float64)
[[0.57351283 0.71070152 2.03907164 1.93622024 2.34373468 1.94435071]]
Iteration 57
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64180829 0.56987047 1.89868868 1.85424464 1.47062079 2.49729969]]
Iteration 58
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.71403008 0.69386538 1.4720356  1.27228906 1.87072287 1.22417188]]
Iteration 59
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.61187365e-183], shape=(1,), dtype=float64)
[[0.64809098 0.66829877 1.10311119 1.48293148 1.54637122 1.20932174]]
Iteration 60
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67725873 0.61089635 1.40989063 2.33256857 1.20190941 1.14124257]]
Iteration 61
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.2616275e-148], shape=(1,), dtype=float64)
[[0.69994554 0.56395375 1.6098966  2.37772298 2.08121816 2.00973279]]
Iteration 62
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.57167335 0.70310369 1.37330904 2.48652124 1.46595536 1.02249459]]
Iteration 63
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.48354152e-214], shape=(1,), dtype=float64)
[[0.70648322 0.59677091 1.39291139 1.51609489 1.73297231 1.76286143]]
Iteration 64
```

```
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.84943962e-69], shape=(1,), dtype=float64)
[[0.55963135 0.71322672 1.00468167 2.45828214 1.89495215 1.87574333]]
Iteration 65
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.76704221e-29], shape=(1,), dtype=float64)
[[0.68943185 0.56340639 1.67408822 2.29861823 2.01583075 2.7121006 ]]
Iteration 66
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6835908  0.71571314 1.13882703 2.15055702 1.57868392 1.88537055]]
Iteration 67
Acquisition function convergence reached at iteration 2.
tf.Tensor([9.63543136e-54], shape=(1,), dtype=float64)
[[0.71336194 0.66353155 2.47494906 1.19132201 2.64988248 1.38944709]]
Iteration 68
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.05347043e-07], shape=(1,), dtype=float64)
[[0.64737299 0.66848237 2.22606006 1.84292391 1.35900373 2.20219953]]
Iteration 69
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.51077494 0.60257915 1.6103297  1.43555831 2.03071031 1.7730185 ]]
Iteration 70
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.01236083e-143], shape=(1,), dtype=float64)
[[0.72498153 0.60450745 1.77088444 2.43828832 1.38984295 2.09044129]]
Iteration 71
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.09468406e-20], shape=(1,), dtype=float64)
[[0.52303092 0.51195317 2.21021907 2.59034319 1.02128151 2.50519273]]
Iteration 72
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.27614499e-129], shape=(1,), dtype=float64)
[[0.61208911 0.67308135 1.04471027 1.50156232 2.06708781 1.33953839]]
Iteration 73
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.5161601  0.57022235 1.24993724 1.84047233 2.06121249 1.04419954]]
Iteration 74
Acquisition function convergence reached at iteration 2.
tf.Tensor([3.53498133e-119], shape=(1,), dtype=float64)
[[0.68386682 0.72848129 2.12971508 2.48151098 1.95376017 1.18253874]]
```

```
Iteration 75
Acquisition function convergence reached at iteration 2.
tf.Tensor([5.49940398e-223], shape=(1,), dtype=float64)
[[0.62684629 0.55419403 1.9415697  1.59044504 2.60494069 1.33750849]]
Iteration 76
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.56114398e-280], shape=(1,), dtype=float64)
[[0.50005942 0.72295068 1.54199199 1.26652111 1.49098418 1.38287504]]
Iteration 77
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50126035 0.5585977  1.69909454 2.34519689 1.28111361 2.56809838]]
Iteration 78
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66150659 0.51293905 1.54195697 1.36950742 2.67983174 1.7085572 ]]
Iteration 79
Acquisition function convergence reached at iteration 2.
tf.Tensor([4.06317034e-151], shape=(1,), dtype=float64)
[[0.55064031 0.60064746 2.6092567  1.40021907 1.23825115 1.6867846 ]]
Iteration 80
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.16035932e-286], shape=(1,), dtype=float64)
[[0.71456832 0.58159229 2.33119218 1.74988509 1.57450596 1.55440498]]
Iteration 81
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59824374 0.52573599 1.00300985 2.3978993  1.51054923 1.65215886]]
Iteration 82
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67463034 0.52670466 1.44923455 1.67606518 1.39907645 2.25064293]]
Iteration 83
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.2380977e-208], shape=(1,), dtype=float64)
[[0.50352274 0.72847731 2.01615005 1.02898396 1.71953218 1.9892621 ]]
Iteration 84
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.55573965 0.55238175 1.92895596 2.18932345 1.13800483 2.21901859]]
Iteration 85
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
```

```
[[0.70984255 0.65561719 2.70080868 1.43102787 1.12781141 1.67623426]]
Iteration 86
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59629159 0.60130497 1.5463238  1.77539371 1.32867759 1.66493707]]
Iteration 87
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.54669663 0.6481768  1.37419386 1.78355939 1.51760595 1.72287587]]
Iteration 88
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.61114508 0.6200288  1.93982118 2.19399919 2.03713086 1.97818798]]
Iteration 89
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59400953 0.53804734 2.30685617 2.40495495 1.35746162 1.22556151]]
Iteration 90
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.62064927 0.54325496 1.38348593 2.01626064 1.01564421 1.17406284]]
Iteration 91
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6215222  0.67993288 1.93141671 1.82100131 2.05604352 1.87522805]]
Iteration 92
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64903595 0.66550485 1.04274101 1.64241606 2.6813848  1.33175652]]
Iteration 93
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.86907818e-153], shape=(1,), dtype=float64)
[[0.50146807 0.58035034 2.34681254 1.1142958  1.70370633 2.39338948]]
Iteration 94
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.5304929  0.70402567 2.13830214 2.29787269 1.30879835 2.60865177]]
Iteration 95
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58854279 0.50816574 2.4267339  2.21931037 1.08745609 2.05953136]]
Iteration 96
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66891647 0.50744954 2.25616392 2.50883524 2.42374051 1.05514312]]
Iteration 97
Acquisition function convergence reached at iteration 2.
tf.Tensor([2.43914214e-61], shape=(1,), dtype=float64)
[[0.67622914 0.61697797 1.00396742 1.16121918 1.02971614 1.02730968]]
Iteration 98
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52666007 0.54301272 1.3474877  1.90620751 1.42875751 1.11935073]]
Iteration 99
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.55138535 0.63713861 1.3417808  1.30634422 2.1190331  1.78348579]]
Iteration 100
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56606805 0.53225103 1.00950307 1.04313015 1.29104506 1.63008696]]
Hyperparameter convergence reached at iteration 8837.
Trained parameters:
amplitude_champ:0 is 41.345

length_scales_champ:0 is [1. 1. 1. 1. 1. 1.]

observation_noise_variance_champ:0 is 48.227

amp_f_mean:0 is 105.347

amp_gamma_L_mean:0 is 0.063

amp_lambda_mean:0 is 84.701

amp_r_mean:0 is 0.037

bias_mean:0 is 0.111

f_tp:0 is 1.345

gamma_L_tp:0 is 0.976

lambda_tp:0 is 0.65

r_tp:0 is 3.093
```

```
Iteration 101
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.32692305e-09], shape=(1,), dtype=float64)
[[0.61695821 0.59754415 1.18920118 1.11292256 1.14553204 2.55922816]]
Iteration 102
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.721155   0.66098572 1.39911967 2.18831101 1.24656877 1.6686223 ]]
Iteration 103
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52510457 0.66140192 2.11012031 1.10506197 1.63735494 1.22581026]]
Iteration 104
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.54844585 0.63288277 2.32449348 1.06445677 1.91477426 2.59939187]]
Iteration 105
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.57086482 0.65080257 1.21878336 1.68899783 1.09659231 1.26999654]]
Iteration 106
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.61925062 0.70873802 1.21734781 1.05265783 1.65126857 1.31746228]]
Iteration 107
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64917881 0.67234714 1.47405655 1.45310681 1.50881588 1.46245527]]
Iteration 108
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64692891 0.71416964 2.46939332 1.95829149 1.47953918 2.08554592]]
Iteration 109
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.568367   0.64644358 1.09962237 2.49158204 1.04169459 2.00248782]]
Iteration 110
Acquisition function convergence reached at iteration 2.
tf.Tensor([1.68426682e-298], shape=(1,), dtype=float64)
[[0.6681928  0.58448135 1.06119237 1.56060107 1.2138251  2.07580997]]
Iteration 111
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.53037398 0.50744933 1.70304657 1.76178576 1.00396754 1.77973803]]
Iteration 112
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72514985 0.58999595 1.4500272  2.31155183 1.85877597 1.053173  ]]
Iteration 113
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66258932 0.52217904 2.00210392 1.14726382 1.55462975 1.25440489]]
Iteration 114
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52675401 0.67167864 1.695539   2.58865365 1.17759902 1.63679067]]
Iteration 115
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59479384 0.69571491 1.70514874 2.22772359 2.43059903 1.06405889]]
Iteration 116
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.55291125 0.63220287 1.68453369 1.25614686 2.69907581 1.27397594]]
Iteration 117
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52280533 0.53766889 1.47543833 2.03306211 2.24070937 1.67434765]]
Iteration 118
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63758078 0.6443794  1.35130894 1.64980806 1.93886558 1.67873966]]
Iteration 119
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.70115901 0.67661648 1.75050821 1.4808034  1.87945124 1.14253245]]
Iteration 120
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.62482814 0.64140418 2.40065664 2.38339661 1.24469608 1.11581052]]
Hyperparameter convergence reached at iteration 6132.
Iteration 121
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6775181  0.66637515 2.30721234 2.46501112 1.38780608 1.71242653]]
```

```
Iteration 122
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.70766557 0.71278274 1.77958176 1.07707942 2.07938426 1.37229075]]
Iteration 123
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.57716786 0.67254129 1.43377544 1.57831405 1.14932725 1.79583918]]
Iteration 124
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.63472909 0.72865581 2.55243163 1.69661853 2.66776602 1.42918674]]
Iteration 125
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.62310963 0.63449175 2.23031317 1.95221165 2.31176422 1.00877193]]
Iteration 126
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64809776 0.55175618 2.47446839 1.29321588 2.16667749 2.4887158 ]]
Iteration 127
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66192605 0.72309321 1.05123606 1.42231027 1.00707657 1.14214201]]
Iteration 128
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56680004 0.54916036 1.11436768 1.38984406 1.79482913 1.13423842]]
Iteration 129
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.65114366 0.61910126 2.58075171 1.10106392 1.59001871 1.56177645]]
Iteration 130
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.53751299 0.53926114 1.54215162 2.09076989 1.92453525 1.40905577]]
Iteration 131
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.54255677 0.61794445 1.62793387 1.67976345 1.03764518 1.62177616]]
Iteration 132
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
```

```
[[0.67880097 0.60009108 2.61431363 1.13533498 2.16648799 1.08217308]]
Iteration 133
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58247662 0.61591859 1.98717078 1.41521809 1.15828178 1.30853843]]
Iteration 134
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.65446827 0.64389148 2.55336575 1.02266382 2.01993886 1.22223972]]
Iteration 135
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69105425 0.64562246 2.14561896 2.13218722 1.23488881 2.16894807]]
Iteration 136
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.64692829 0.53408344 1.9704271  1.45405571 1.41276656 1.55671281]]
Iteration 137
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.62892477 0.65442559 2.14098394 2.07757094 2.12190626 1.42524019]]
Iteration 138
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52024133 0.58558901 1.86900222 1.88463981 1.20743008 1.67330726]]
Iteration 139
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72355553 0.69133652 2.5038642  2.11710489 2.47256497 1.28595685]]
Iteration 140
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.53471663 0.51172168 2.12524576 2.52919697 1.33192507 1.72292844]]
Iteration 141
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56410127 0.53536935 1.23085027 1.01361128 2.37664316 1.89382558]]
Iteration 142
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69697291 0.72717714 1.55293697 1.14727289 2.57165152 1.27357025]]
Iteration 143
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59630671 0.6732301  1.60621251 1.51549166 1.80850155 2.22820623]]
Iteration 144
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72396328 0.55281922 1.24916843 2.23291534 1.54595989 1.64112933]]
Iteration 145
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.61566788 0.53621621 2.15370679 1.74350078 1.1811509  2.56090474]]
Iteration 146
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68971692 0.69607144 1.74399273 1.11273433 1.74948249 1.18074136]]
Iteration 147
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.62060803 0.63887181 1.17273077 2.61663311 2.62568248 1.15276604]]
Iteration 148
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.66534602 0.54036786 2.58220504 2.36025551 1.32905312 2.37681623]]
Iteration 149
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68485714 0.54590449 1.51058126 1.25672139 1.0059925  1.95602472]]
Iteration 150
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60690266 0.65050802 1.08381416 1.63142912 1.07689989 2.01799761]]
Trained parameters:
amplitude_champ:0 is 11.105

length_scales_champ:0 is [0.156 1.    1.    1.    1.    1.   ]

observation_noise_variance_champ:0 is 81.696

amp_f_mean:0 is 29.163

amp_gamma_L_mean:0 is 20.473

amp_lambda_mean:0 is 131.732
```

```
amp_r_mean:0 is 10.716

bias_mean:0 is 1.241

f_tp:0 is 0.407

gamma_L_tp:0 is 0.544

lambda_tp:0 is -0.136

r_tp:0 is 2.387

Iteration 151
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56119571 0.51501265 2.00664096 2.40498653 1.09396147 1.13642423]]
Iteration 152
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.71367833 0.50632805 2.0917157  2.32382413 1.18150183 1.58287867]]
Iteration 153
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68166036 0.55025662 1.59287863 2.44147038 2.15230673 2.34549476]]
Iteration 154
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58428579 0.57686974 1.35555186 2.15373414 1.11305829 2.23876783]]
Iteration 155
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50358522 0.61392509 1.03399593 1.3573631  2.14750361 1.30719731]]
Iteration 156
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58270065 0.59893003 2.15353964 1.46163102 1.77823534 2.51831423]]
Iteration 157
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60142189 0.59965102 1.64317567 2.14281638 1.82337593 1.12982756]]
Iteration 158
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
```

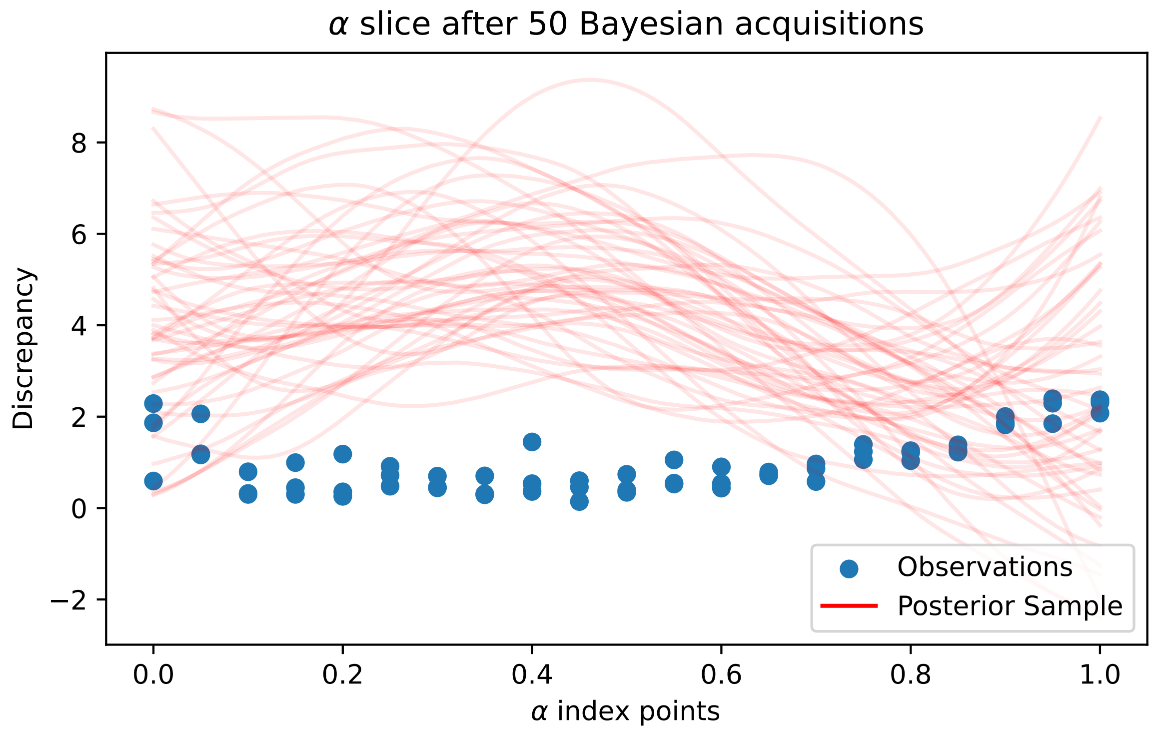
```
[[0.54081119 0.57452521 1.382596   1.42997758 1.9381712  2.30170549]]
Iteration 159
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58972774 0.7076727  1.21558627 2.45874422 1.46492863 1.14354154]]
Iteration 160
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6228606  0.68004748 1.32462486 1.98032554 1.88424346 2.49798528]]
Iteration 161
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.65326561 0.68625251 2.32308365 1.75915477 1.90758743 2.41723462]]
Iteration 162
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58776029 0.56772892 2.37744742 1.93783203 1.7259168  1.47722828]]
Iteration 163
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.58532385 0.55703414 1.46450602 1.23884018 1.31744424 1.70081866]]
Iteration 164
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68996135 0.72928376 2.04427258 1.9684908  1.62186799 1.3977521 ]]
Iteration 165
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.51866861 0.67904776 1.51807385 2.38464639 1.42120956 1.76965015]]
Iteration 166
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50311825 0.61968263 2.35455187 1.75587384 1.03733024 1.03366824]]
Iteration 167
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50037547 0.51862972 1.0785204  1.09191937 2.32818764 1.29973719]]
Iteration 168
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68459072 0.50294195 2.23909455 1.40588373 1.03450806 2.59758803]]
Iteration 169
Acquisition function convergence reached at iteration 2.
```

```
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.71541474 0.59747872 1.49219465 1.74633679 1.87063082 2.20697312]]
Iteration 170
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72050389 0.56986706 2.53871603 1.32243899 2.29416984 2.09223566]]
Iteration 171
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56922286 0.5237811  2.3235421  1.07928989 1.64456642 1.16443357]]
Iteration 172
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6092477  0.56860459 1.7257988  2.5992863  2.5029684  1.32554404]]
Iteration 173
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72573697 0.70099093 2.26254358 2.45336831 2.60326875 1.25631391]]
Iteration 174
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67204802 0.59665187 2.13602897 2.18244742 2.37227894 1.36384019]]
Iteration 175
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.57272972 0.68330551 1.40240445 2.49242177 2.0902923  2.20634142]]
Iteration 176
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.51033728 0.57926216 2.01544092 1.96437751 1.10226157 1.53387695]]
Iteration 177
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6765223  0.63703115 1.88869723 1.7274064  2.34537034 1.24610503]]
Iteration 178
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69016224 0.71215371 2.65977052 1.1381246  1.26380776 1.07905045]]
Iteration 179
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.59428142 0.65489498 1.07108063 1.75580122 1.64936038 1.70628756]]
Iteration 180
```

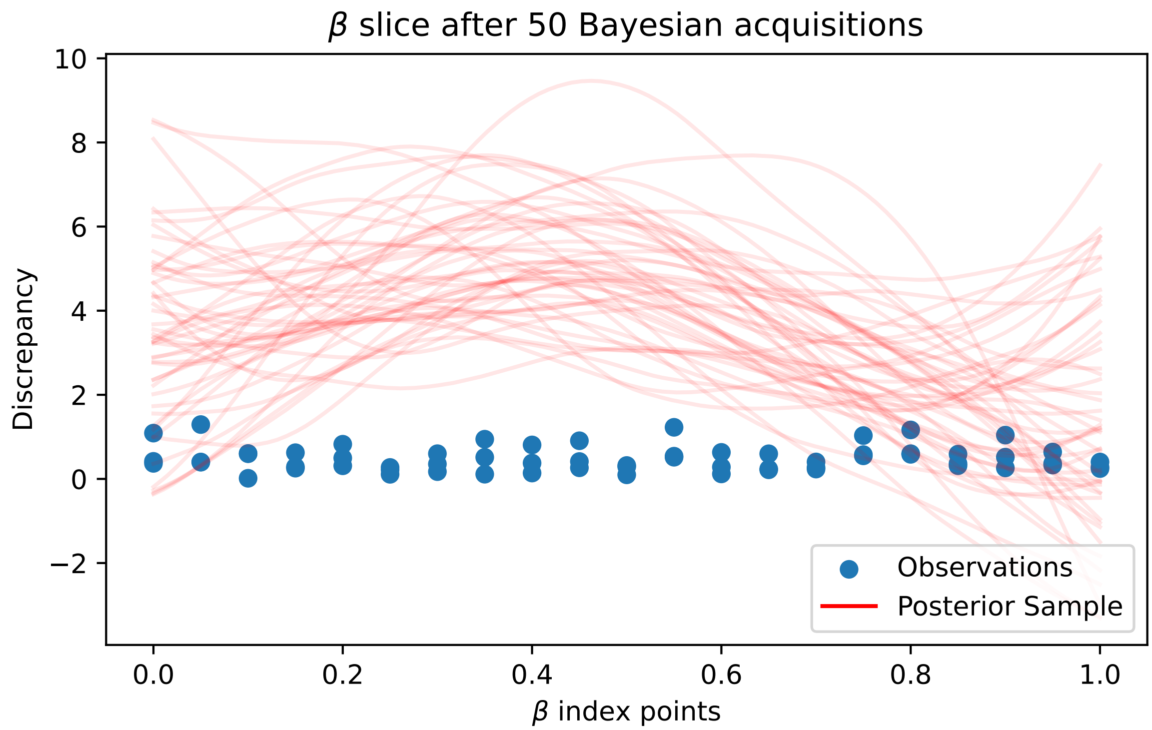
```
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.7067939  0.70444474 1.20725033 1.68506944 1.08961179 1.07957785]]
Hyperparameter convergence reached at iteration 8886.
Iteration 181
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52622324 0.58471207 2.34354419 2.32533649 1.89635406 2.05582821]]
Iteration 182
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.68260461 0.51753981 2.23985064 2.53873259 2.42422938 2.12798942]]
Iteration 183
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6503292  0.51408819 2.70642754 2.23914021 2.6487956  1.50735469]]
Iteration 184
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.61648297 0.70677459 1.60877013 2.28494925 2.4758276  1.71184165]]
Iteration 185
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60476787 0.55169771 1.55013859 2.67609734 1.63589573 2.07965495]]
Iteration 186
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50671136 0.72738819 1.75142369 1.5617947  1.61278575 2.25113689]]
Iteration 187
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.60639185 0.60312703 1.22275968 1.53174542 1.74011487 1.26078777]]
Iteration 188
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.54815092 0.60251661 1.34551975 2.39501615 2.39649869 1.47359402]]
Iteration 189
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.65812474 0.64859718 1.65827196 2.29261563 2.67931798 2.39759827]]
Iteration 190
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
```

```
[[0.62405743 0.62094067 1.24609737 2.21885769 1.79496094 1.67864332]]
Iteration 191
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.67358057 0.53088107 1.08557358 1.01701522 1.70879107 1.64703741]]
Iteration 192
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.57895428 0.71441273 1.38204916 1.30839116 1.59294413 1.64534571]]
Iteration 193
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.52610589 0.58295747 1.88028905 1.16178983 1.39868976 1.35681787]]
Iteration 194
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.72404034 0.614805   2.56598478 1.94646823 1.69754879 1.86639447]]
Iteration 195
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.6274797  0.61707298 1.73890593 2.43655097 1.52380751 1.14886711]]
Iteration 196
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.50790816 0.5705414  2.02785319 1.22275354 2.6188125  1.40441598]]
Iteration 197
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.54046619 0.56989415 2.21098093 1.79372156 1.57771959 1.15243442]]
Iteration 198
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.51546841 0.69941296 1.4631154  1.18804609 1.7272786  1.90894337]]
Iteration 199
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.69920429 0.70092483 1.88751761 2.64235821 1.34328646 2.05161393]]
Iteration 200
Acquisition function convergence reached at iteration 2.
tf.Tensor([0.], shape=(1,), dtype=float64)
[[0.56576197 0.5403351  2.46790037 2.6278893  1.47525788 1.71713282]]
Hyperparameter convergence reached at iteration 9155.
Trained parameters:
```

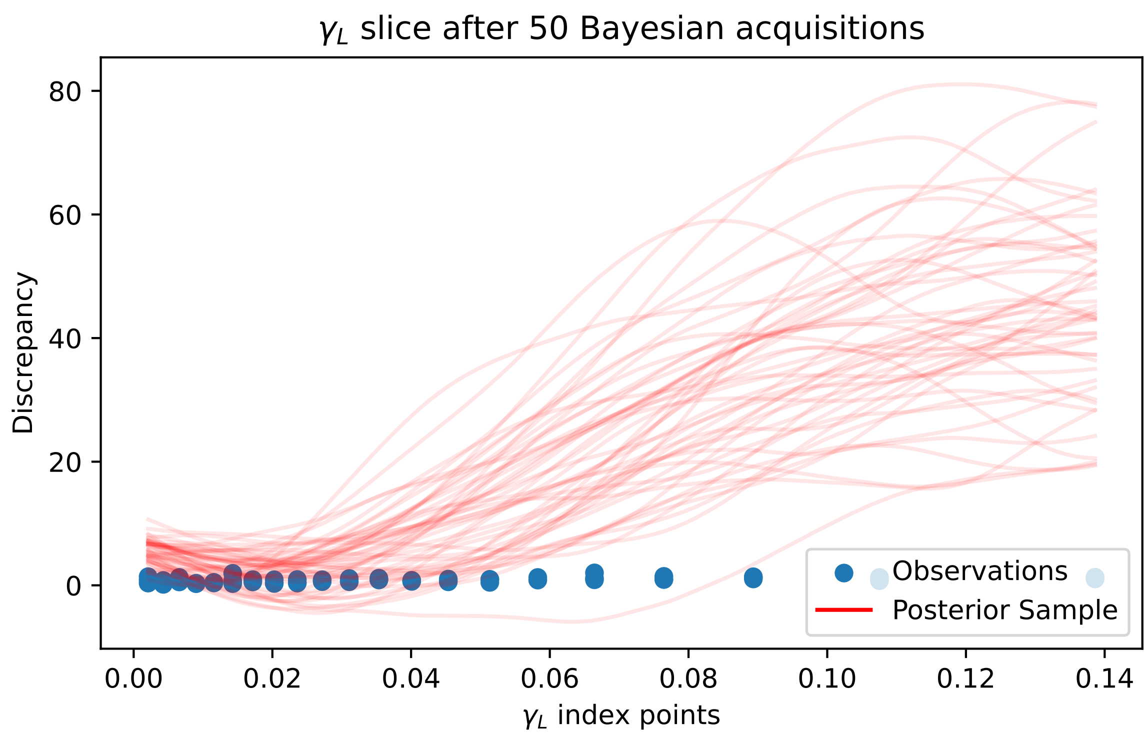
```
amplitude_champ:0 is 12.672

length_scales_champ:0 is [0.188 1.    1.    1.    1.    1.   ]

observation_noise_variance_champ:0 is 78.256

amp_f_mean:0 is 76.07

amp_gamma_L_mean:0 is 38.623

amp_lambda_mean:0 is 135.88

amp_r_mean:0 is 0.213

bias_mean:0 is 2.671

f_tp:0 is 0.824

gamma_L_tp:0 is 0.795

lambda_tp:0 is -0.049

r_tp:0 is 1.921

[0.65446827 0.64389148 2.55336575 1.02266382 2.01993886 1.22223972]
[0.67204802 0.59665187 2.13602897 2.18244742 2.37227894 1.36384019]
[0.60639185 0.60312703 1.22275968 1.53174542 1.74011487 1.26078777]
[0.72404034 0.614805   2.56598478 1.94646823 1.69754879 1.86639447]
[0.51546841 0.69941296 1.4631154  1.18804609 1.7272786  1.90894337]
[0.69920429 0.70092483 1.88751761 2.64235821 1.34328646 2.05161393]
[0.56576197 0.5403351  2.46790037 2.6278893  1.47525788 1.71713282]
[0.56576197 0.5403351  2.46790037 2.6278893  1.47525788 1.71713282]
```



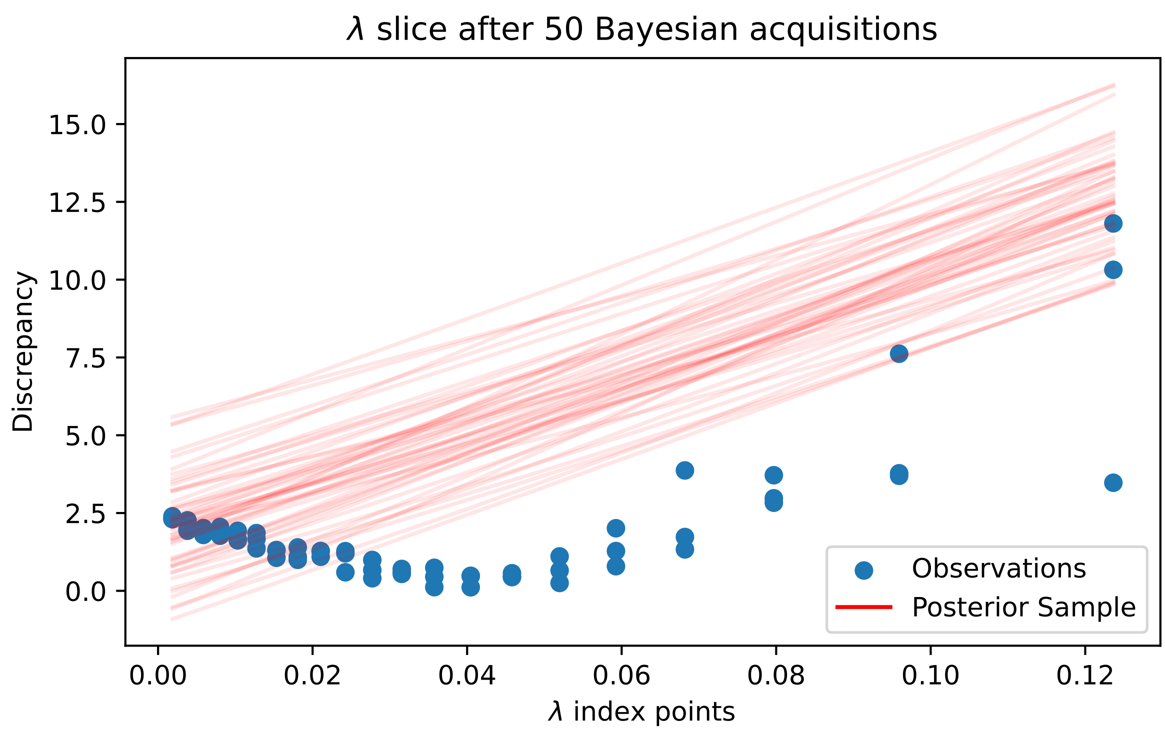
α slice after 50 Bayesian acquisitions
Discrepancy
8
6
4
2
0
−2
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample



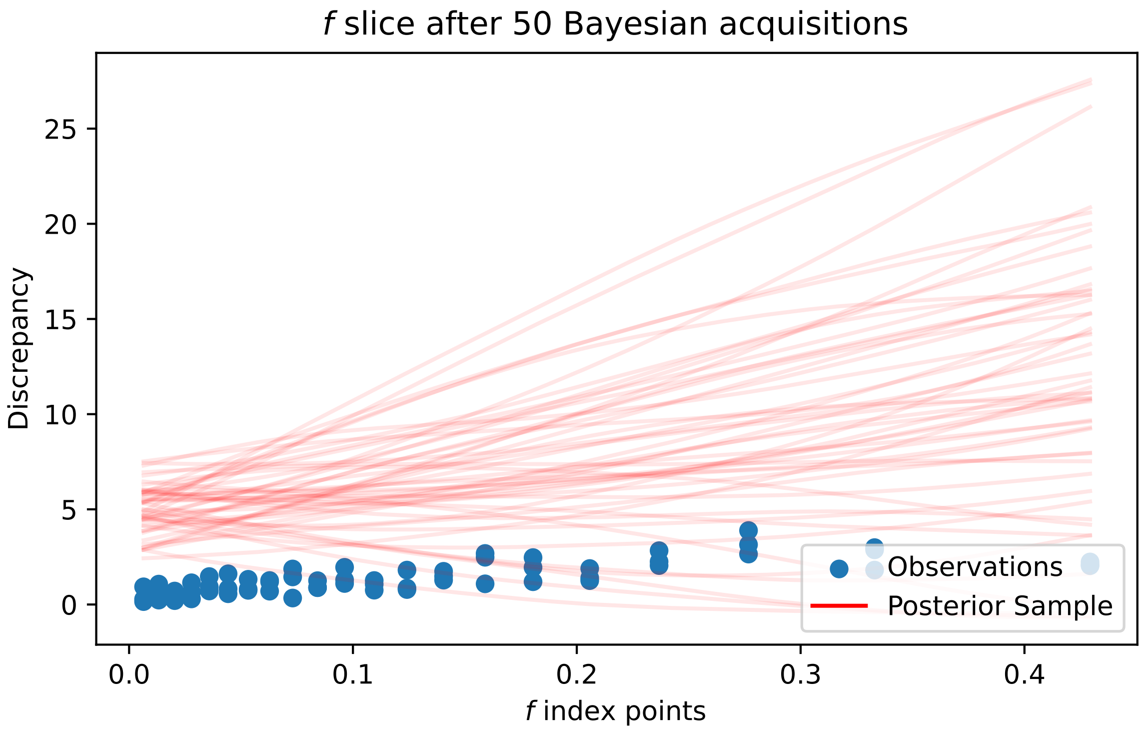
β slice after 50 Bayesian acquisitions
Discrepancy
10
8
6
4
2
0
−2
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

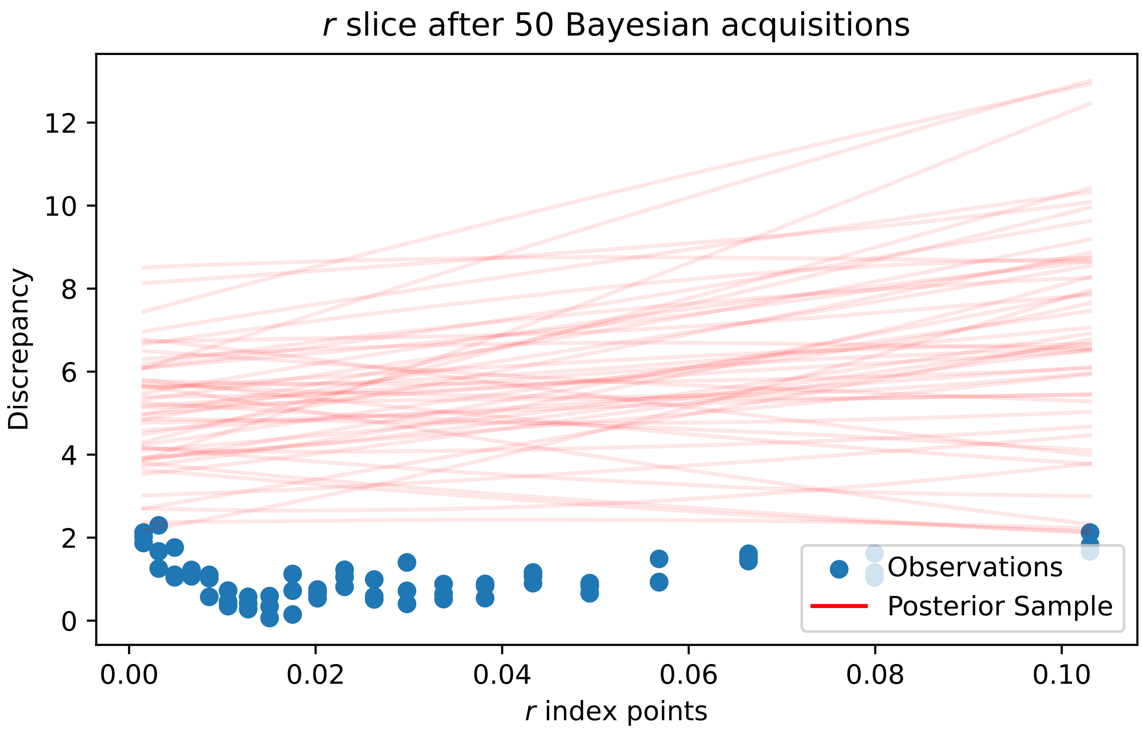


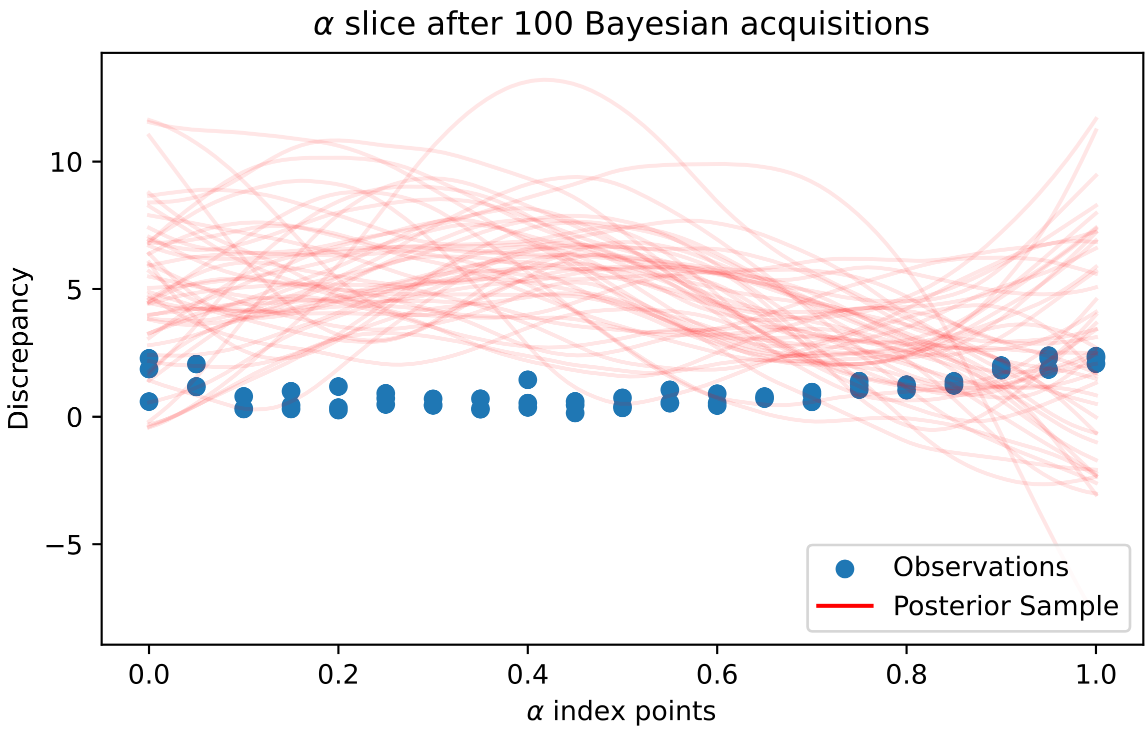
γ_L slice after 50 Bayesian acquisitions
Discrepancy
γ_L index points
Observations
Posterior Sample

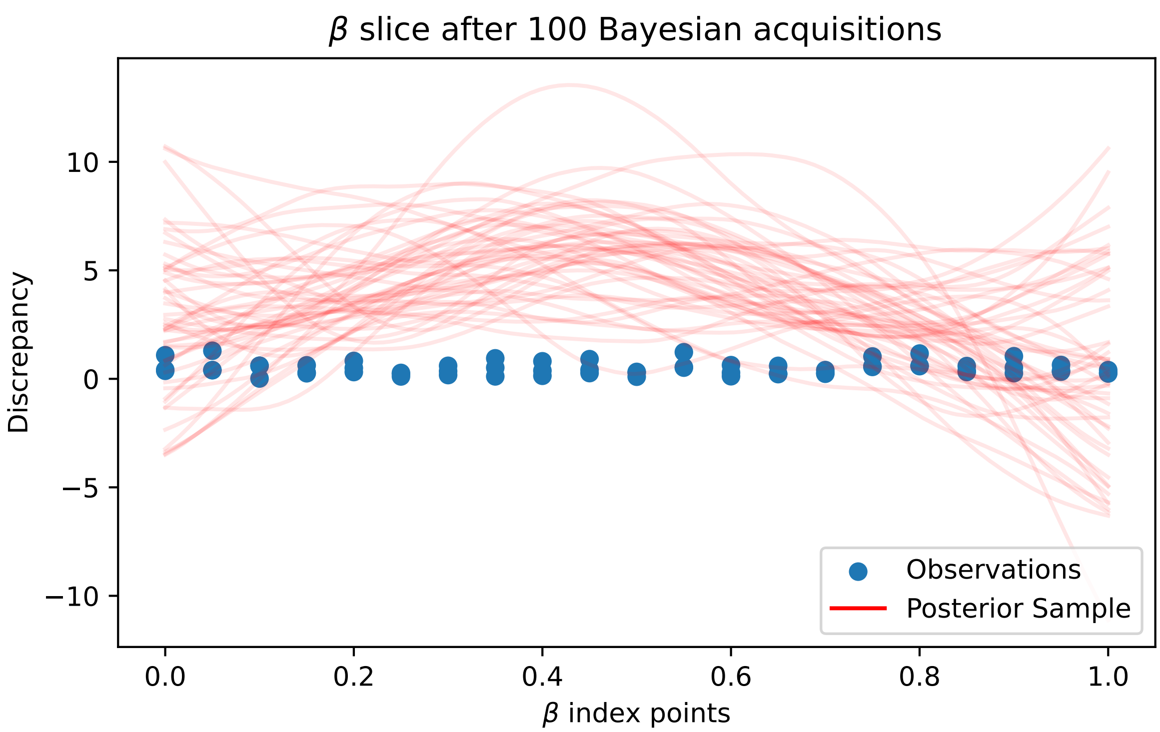


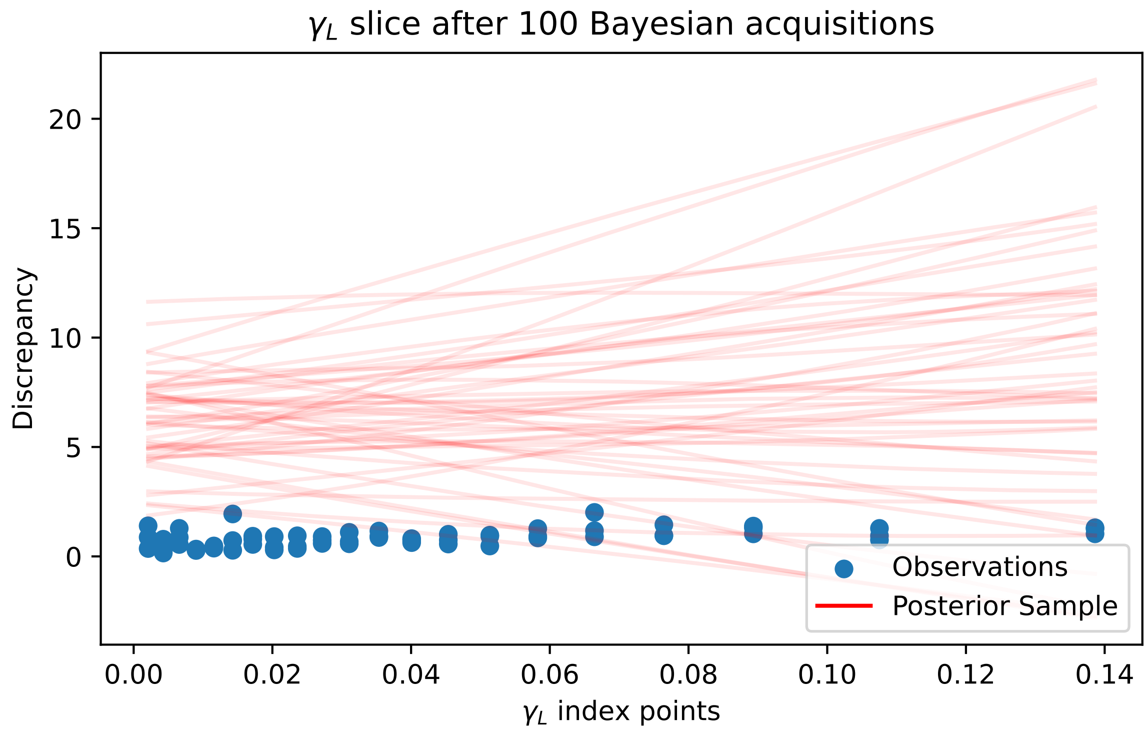
λ slice after 50 Bayesian acquisitions
Discrepancy
λ index points
Observations
Posterior Sample


f slice after 50 Bayesian acquisitions
Discrepancy
f index points
Observations
Posterior Sample
0
5
10
15
20
25
0.0
0.1
0.2
0.3
0.4

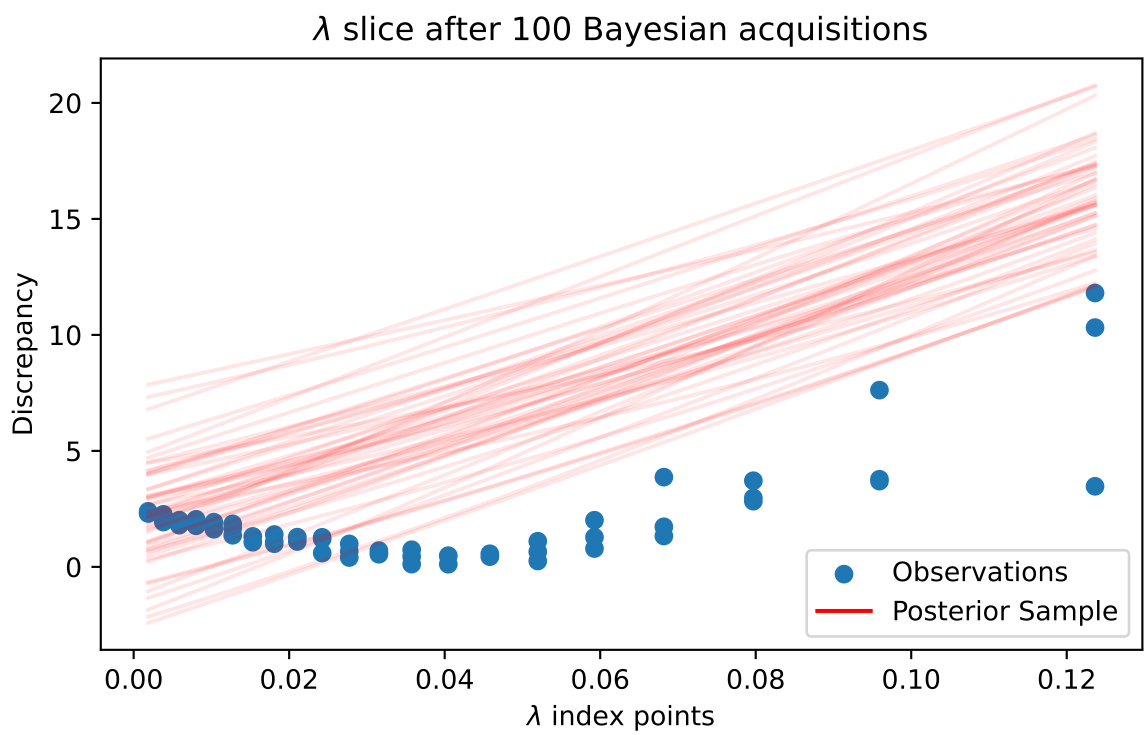

r slice after 50 Bayesian acquisitions
Discrepancy
r index points
Observations
Posterior Sample
0
2
4
6
8
10
12
0.00
0.02
0.04
0.06
0.08
0.10


α slice after 100 Bayesian acquisitions
Discrepancy
10
5
0
−5
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample

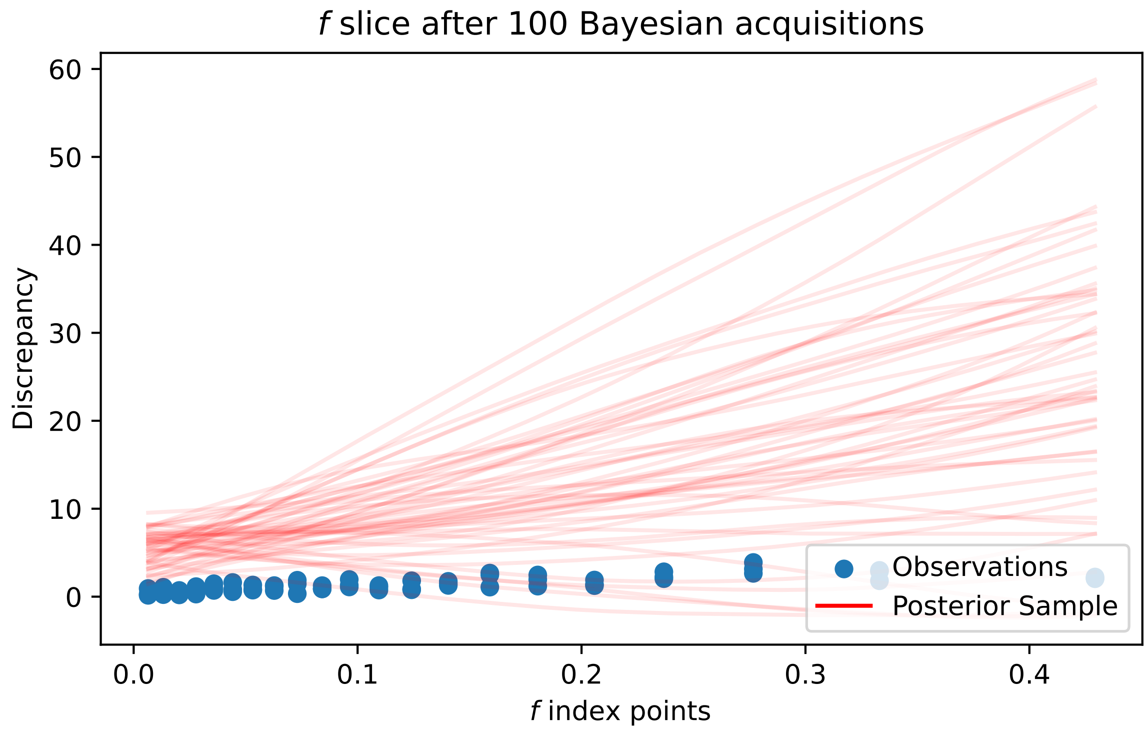

β slice after 100 Bayesian acquisitions
Discrepancy
10
5
0
−5
−10
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

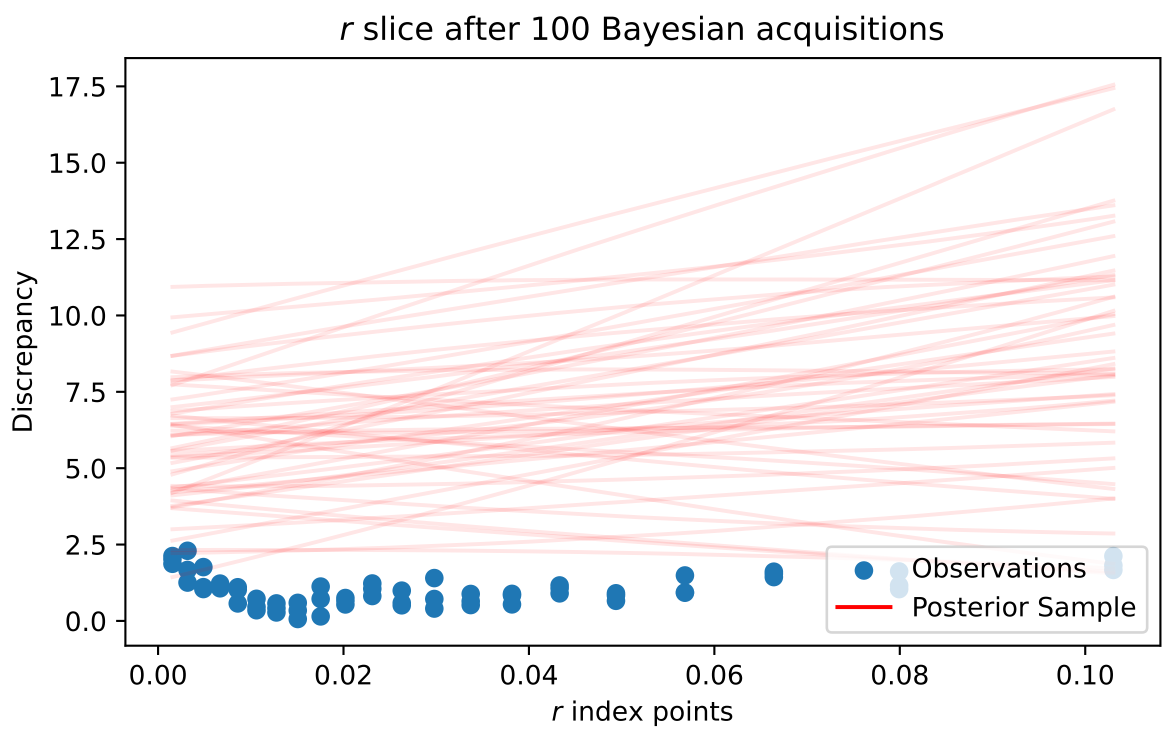


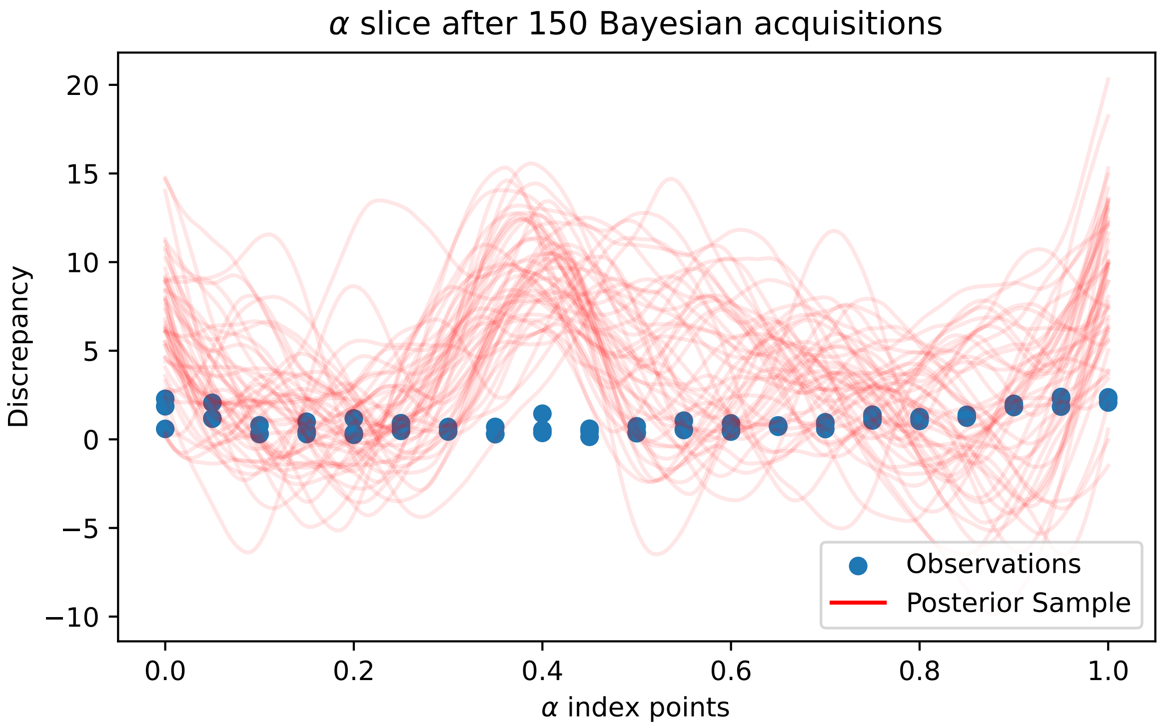
γ_L slice after 100 Bayesian acquisitions
Discrepancy
γ_L index points
Observations
Posterior Sample



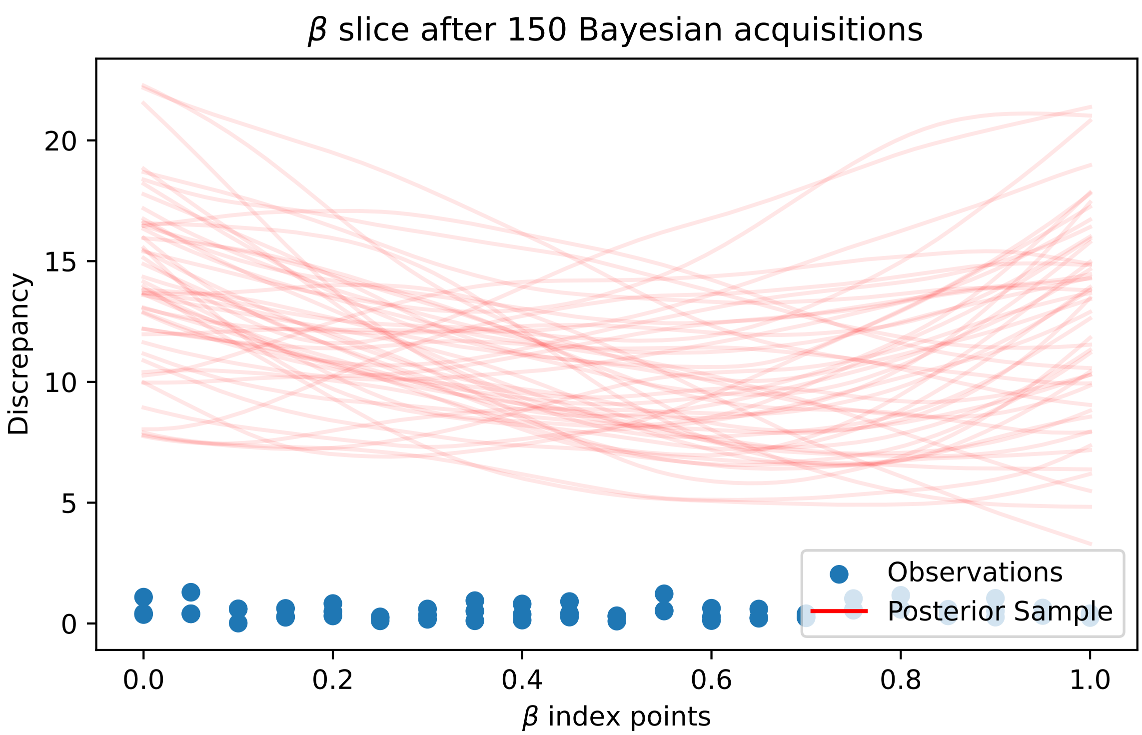
λ slice after 100 Bayesian acquisitions
Discrepancy
λ index points
Observations
Posterior Sample


f slice after 100 Bayesian acquisitions
Discrepancy
f index points
Observations
Posterior Sample
60
50
40
30
20
10
0
0.0
0.1
0.2
0.3
0.4

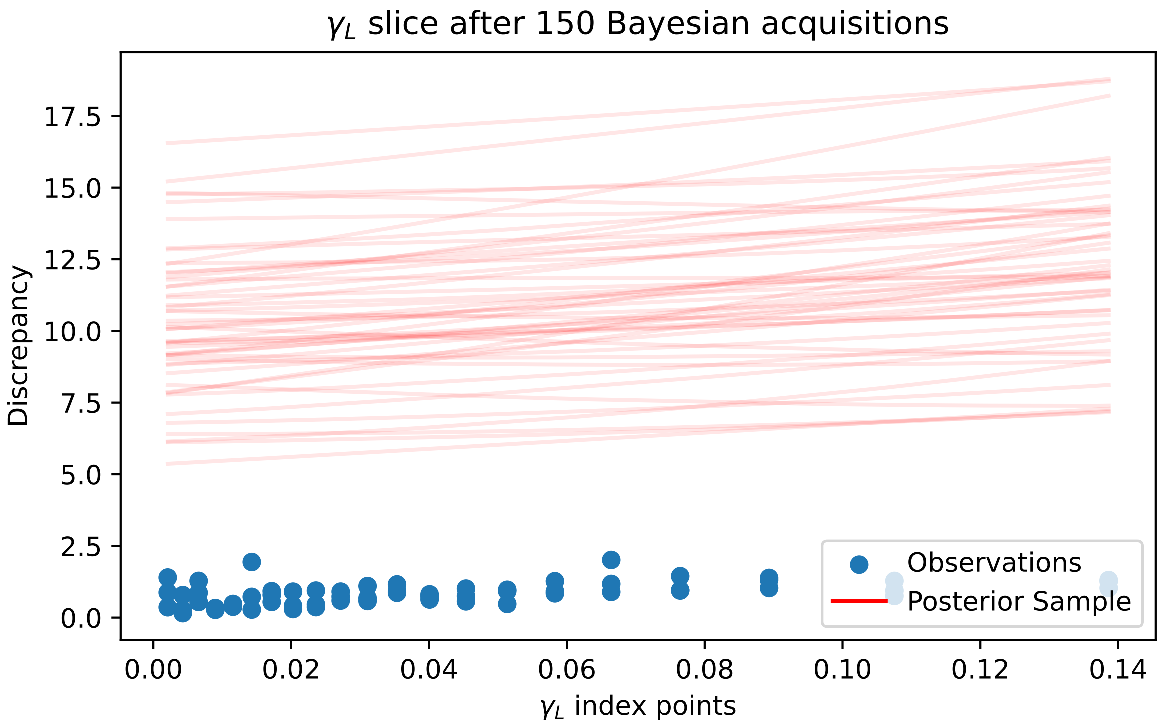

r slice after 100 Bayesian acquisitions
Discrepancy
r index points
Observations
Posterior Sample
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.00
0.02
0.04
0.06
0.08
0.10

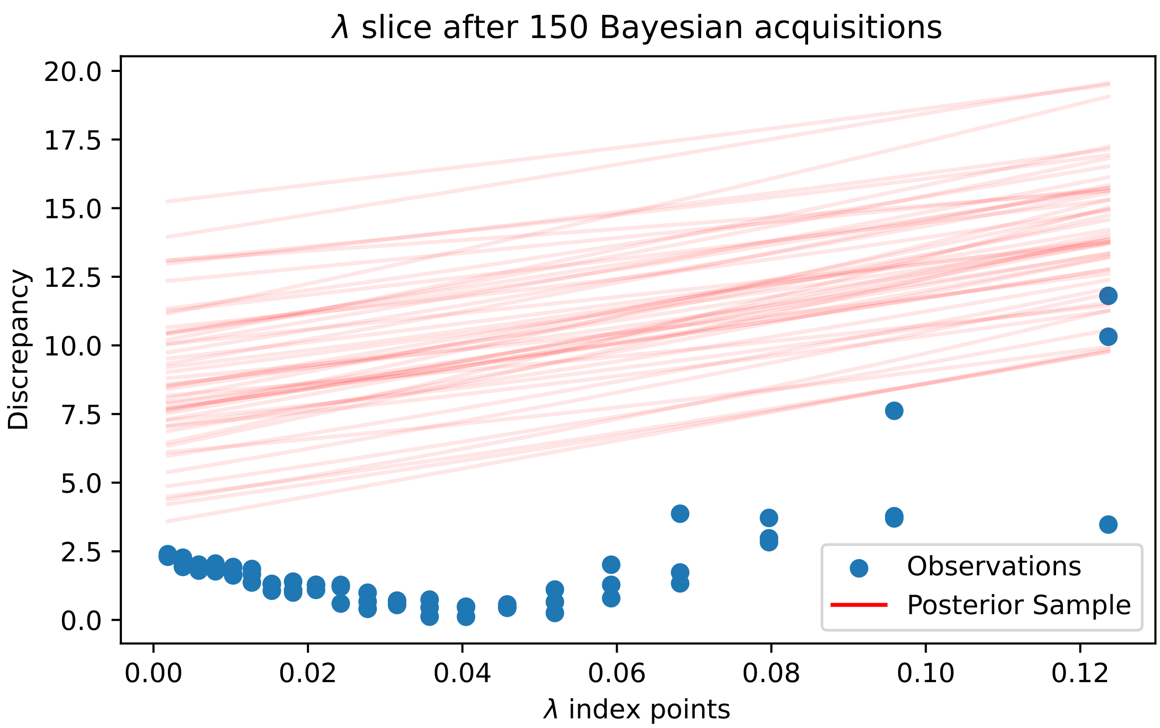


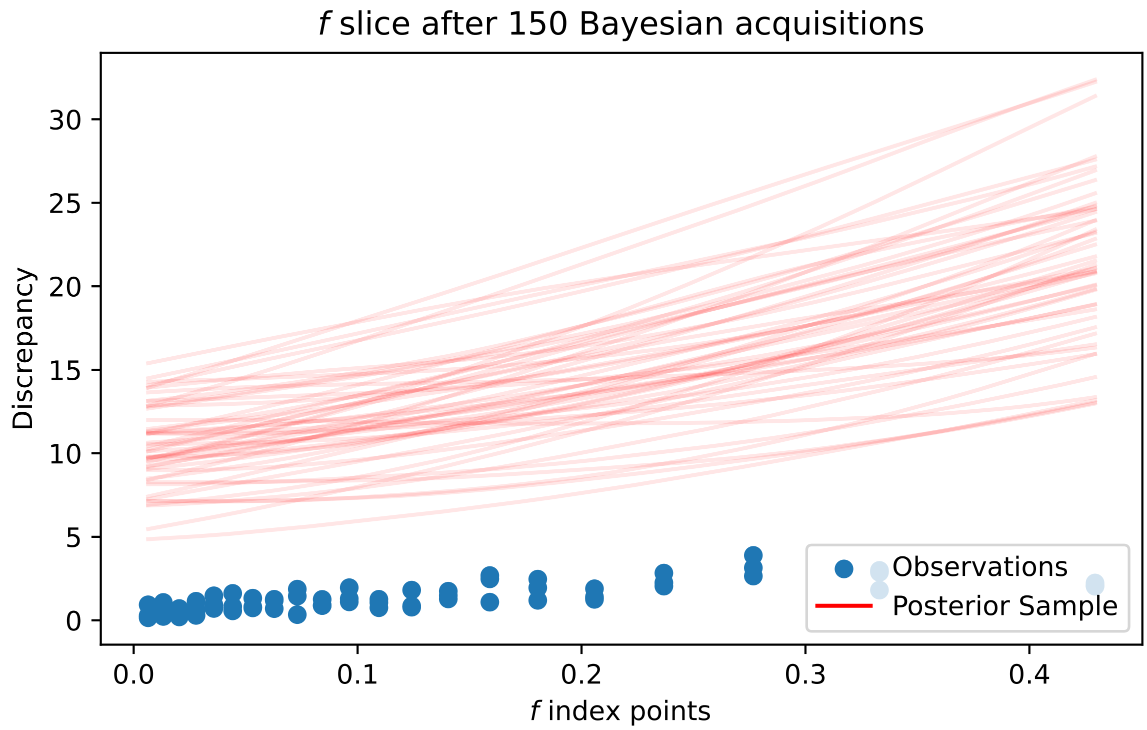
α slice after 150 Bayesian acquisitions
Discrepancy
20
15
10
5
0
−5
−10
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample



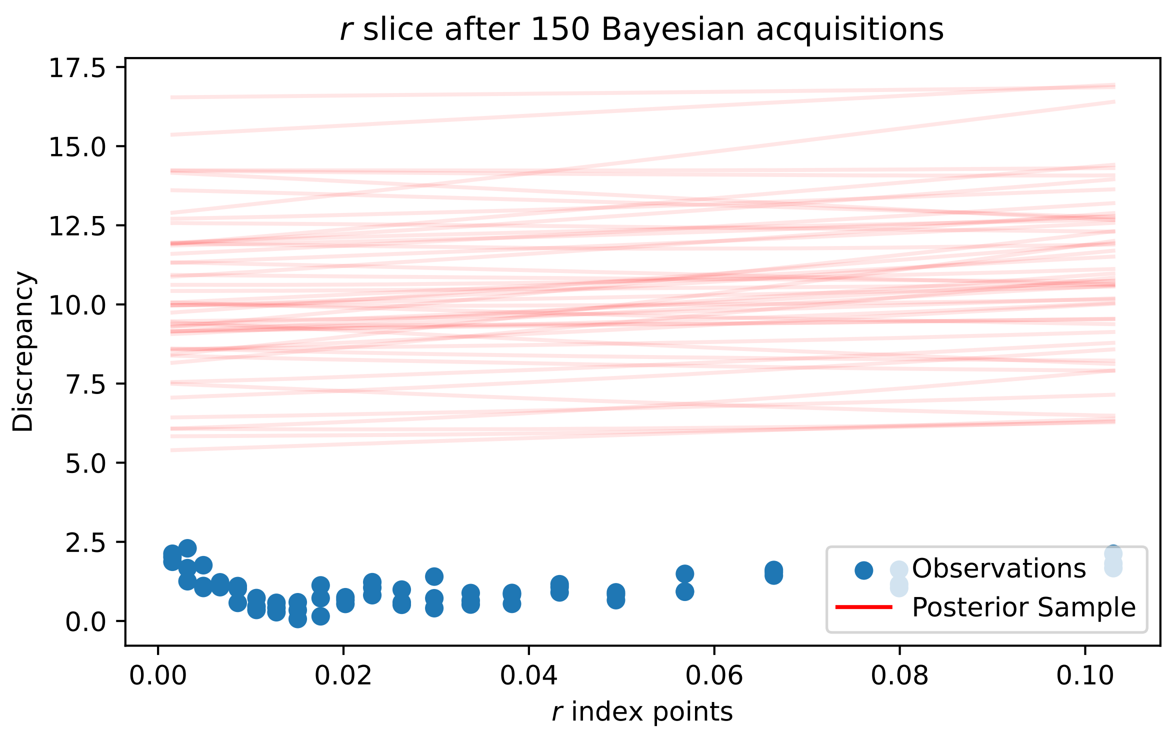
β slice after 150 Bayesian acquisitions
Discrepancy
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample


γ_L slice after 150 Bayesian acquisitions
Discrepancy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γ_L index points
Observations
Posterior Sample

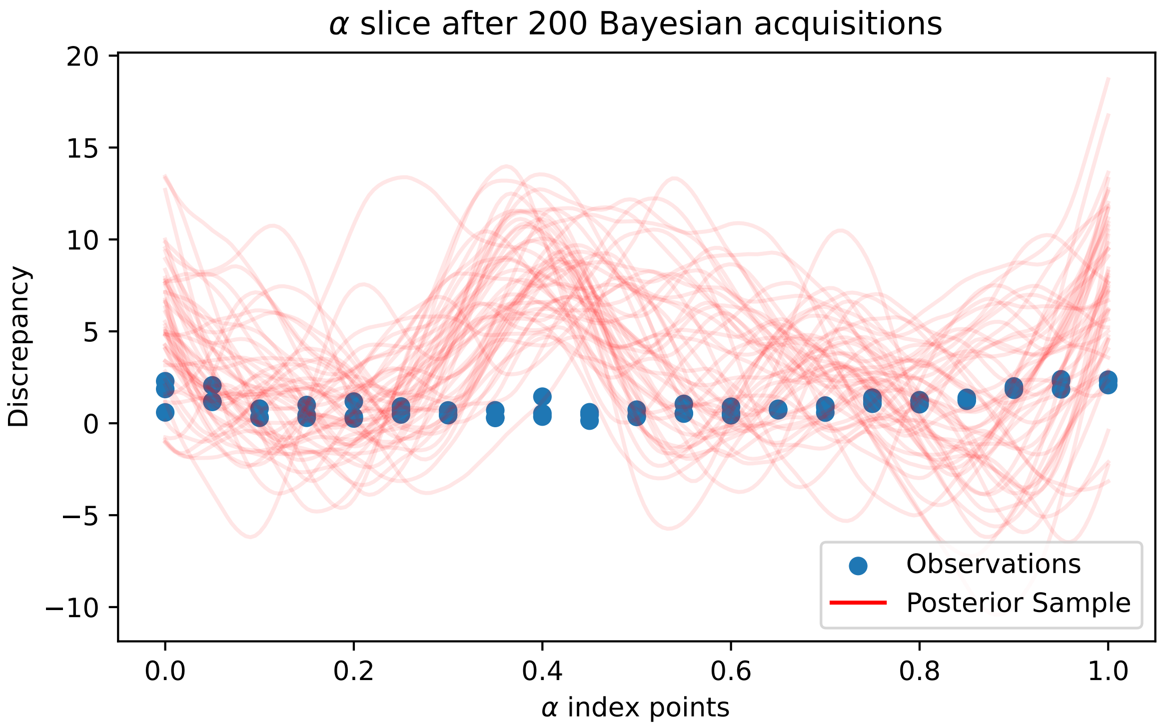

λ slice after 150 Bayesian acquisitions
Discrepancy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample



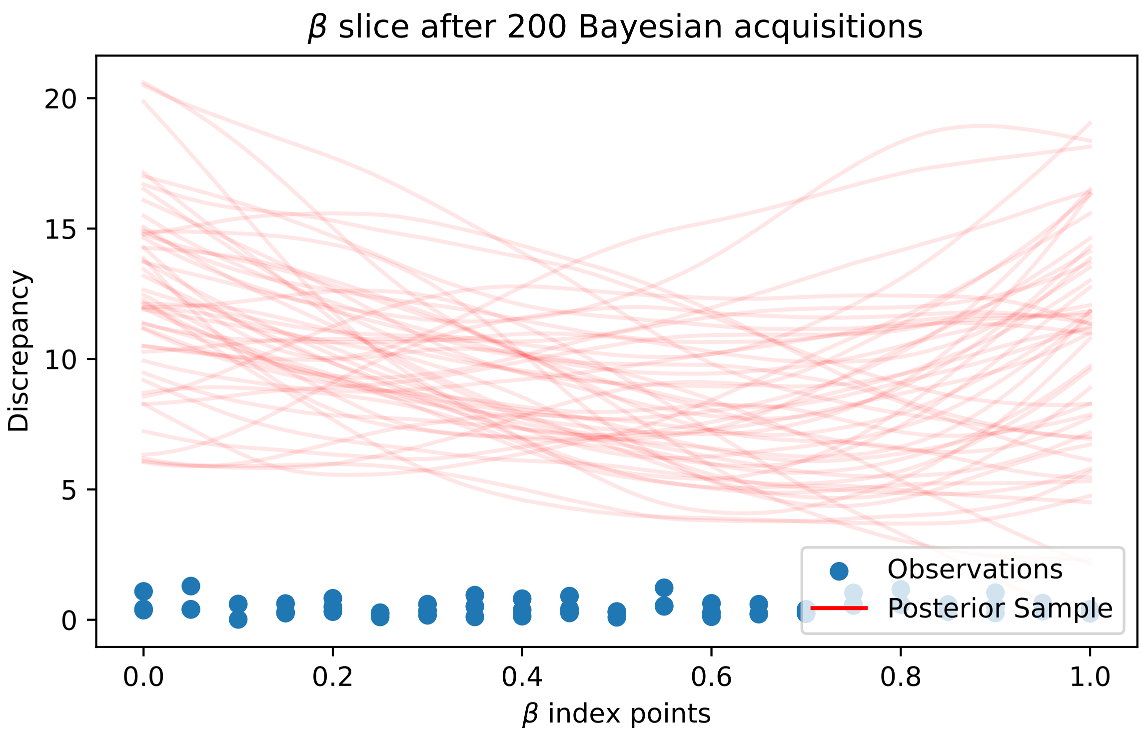
f slice after 150 Bayesian acquisitions
Discrepancy
f index points
Observations
Posterior Sample
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4



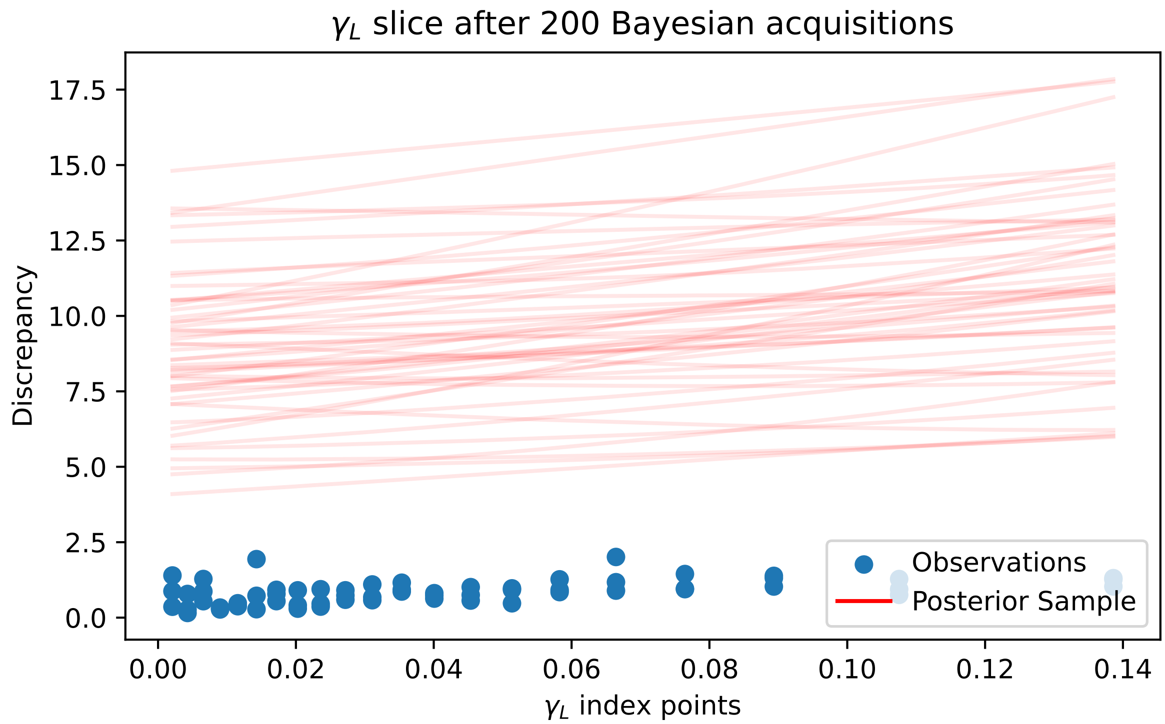
r slice after 150 Bayesian acquisitions
Discrepancy
r index points
Observations
Posterior Sample
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.00
0.02
0.04
0.06
0.08
0.10

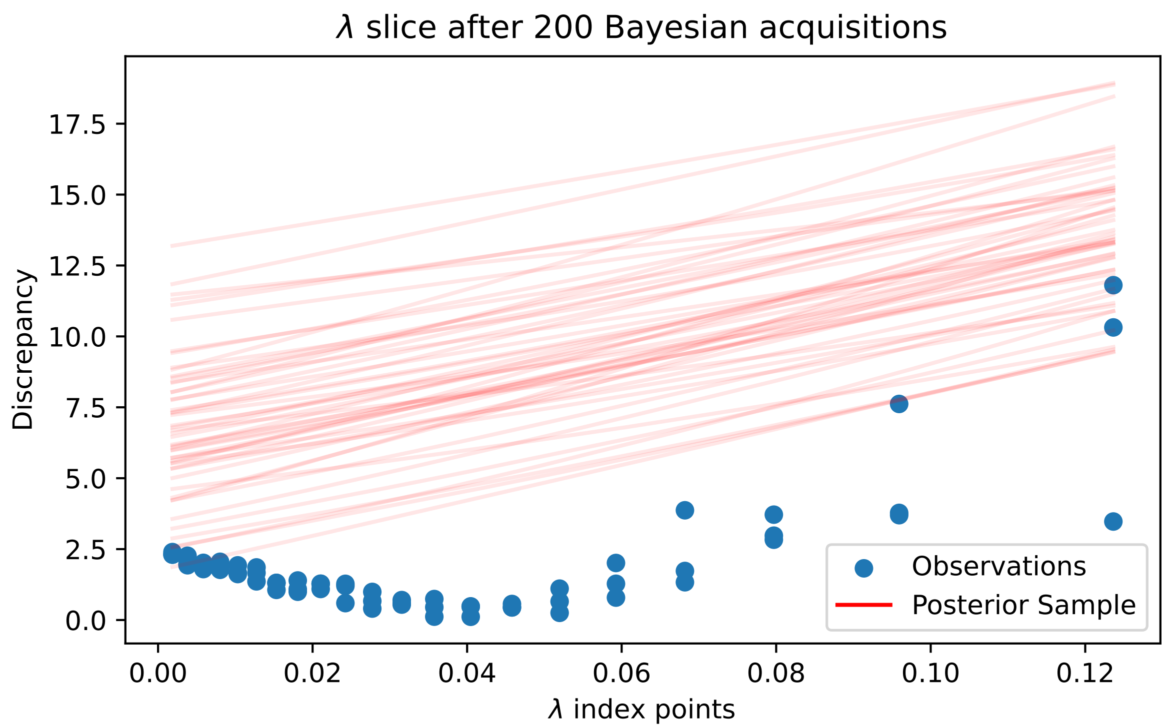


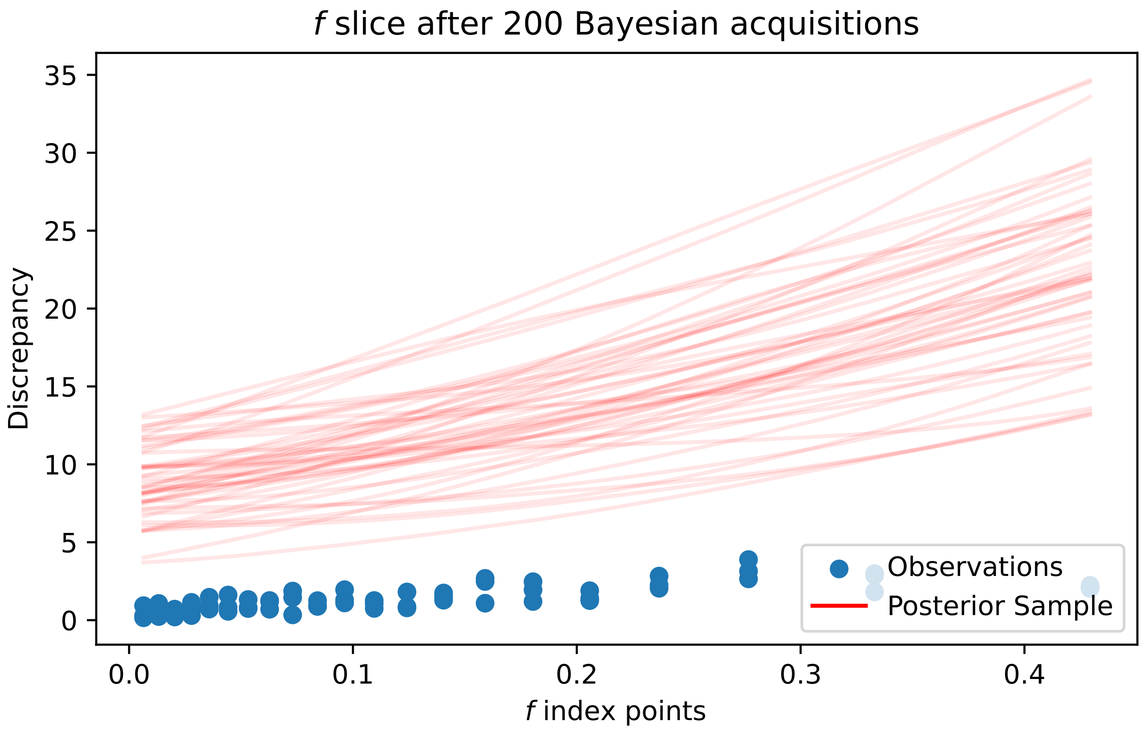
α slice after 200 Bayesian acquisitions
Discrepancy
α index points
Observations
Posterior Sample

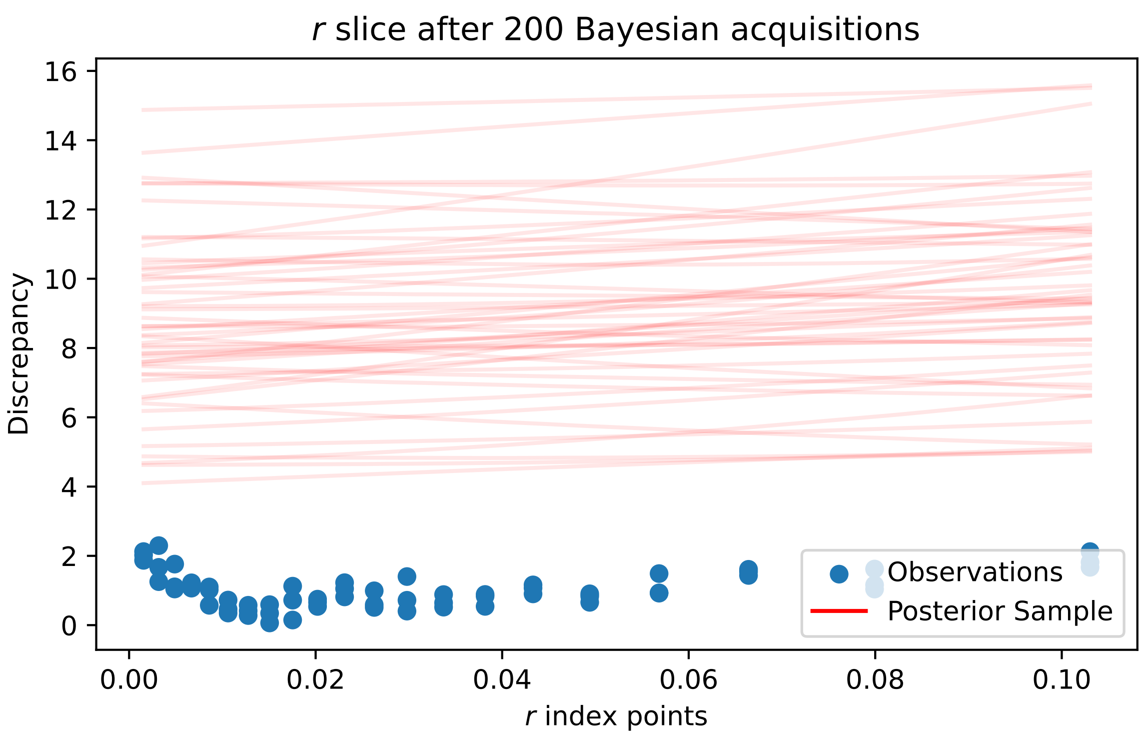


β slice after 200 Bayesian acquisitions
Discrepancy
β index points
Observations
Posterior Sample


γ_L slice after 200 Bayesian acquisitions
Discrepancy
γ_L index points
Observations
Posterior Sample


λ slice after 200 Bayesian acquisitions
Discrepancy
λ index points
Observations
Posterior Sample


f slice after 200 Bayesian acquisitions
Discrepancy
f index points
Observations
Posterior Sample
0
5
10
15
20
25
30
35
0.0
0.1
0.2
0.3
0.4


r slice after 200 Bayesian acquisitions
Discrepancy
r index points
Observations
Posterior Sample
0
2
4
6
8
10
12
14
16
0.00
0.02
0.04
0.06
0.08
0.10