Inference on the Champagne Model using a Gaussian Process

TODO

- Change outputs

Setting up the Champagne Model

Imports

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

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

gpu_devices = tf.config.experimental.list_physical_devices("GPU")
```

```
for device in gpu_devices:
    tf.config.experimental.set_memory_growth(device, True)
```

```
2024-05-14 18:23:54.782528: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensor
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-05-14 18:23:55.442014: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Wa
2024-05-14 18:23:57.939382: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:9
2024-05-14 18:23:57.978066: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot d
Skipping registering GPU devices...
```

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

gamma_L_max = 1/30
lambda_max = 0.1
f_max = 1/14
r_max = 1/14

num_lhc_samples = 36
initial_repeats = 1
```

```
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"

    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,
```

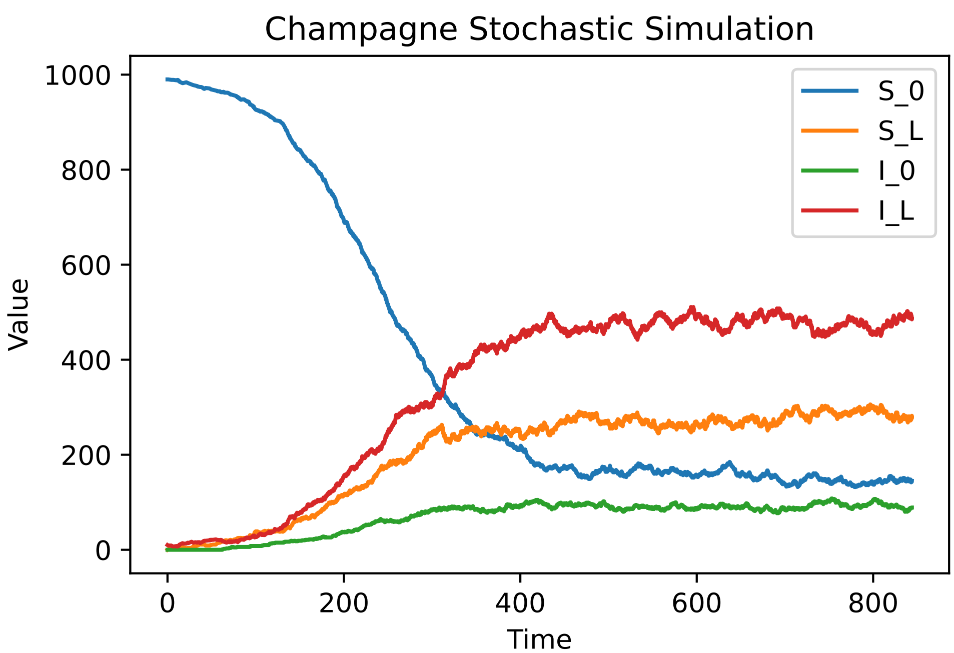
```
                "I_0": I_0,
                "I_L": I_L + 1,
                "inc_counter": inc_counter + silent_incidences + 1,
            },
            {
                "t": t,
                "S_0": S_0 - 1,
                "S_L": S_L + 1,
                "I_0": I_0,
                "I_L": I_L,
                "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"]
```

```python
    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, mean_of = 30):  # best is L1 norm
    mean_obs = 0
    for i in range(mean_of):
        x = champ_sum_stats(alpha_, beta_, gamma_L, lambda_, f, r)
        mean_obs += (
            1
            / mean_of
            * np.log(np.linalg.norm((x - observed_sum_stats) / observed_sum_stats))
        )
    # return np.sum(np.abs((x - observed_sum_stats) / observed_sum_stats))
    # return np.linalg.norm((x - observed_sum_stats) / observed_sum_stats)
    return mean_obs
```

Gaussian Process Regression on Final Prevalence Discrepency

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

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

LHC_sampler = qmc.LatinHypercube(d=6, seed=my_seed)
LHC_samples = LHC_sampler.random(n=num_lhc_samples)

# Using Champagne Initialisation table 2
LHC_samples[:, 2] = gamma_L_max * LHC_samples[:, 2]
LHC_samples[:, 3] = lambda_max * LHC_samples[:, 3]
LHC_samples[:, 4] = f_max * LHC_samples[:, 4]
LHC_samples[:, 5] = r_max * LHC_samples[:, 5]


# LHC_samples[:, 2] = 1/50* LHC_samples[:, 2]
# LHC_samples[:, 3] = 0.2 * LHC_samples[:, 3]
# LHC_samples[:, 4] = 1/10 * LHC_samples[:, 4]
# LHC_samples[:, 5] = 1/10 * LHC_samples[:, 5]
# 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, initial_repeats, axis = 0)

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

print(LHC_indices_df.head())
```

```
      alpha      beta   gamma_L    lambda         f         r
0  0.638900  0.614374  0.021761  0.039933  0.003810  0.007869
1  0.276701  0.070771  0.031115  0.085963  0.050461  0.070414
2  0.727164  0.756949  0.001619  0.064036  0.011960  0.001591
3  0.155333  0.292447  0.004117  0.048578  0.027027  0.020526
4  0.181960  0.003381  0.018591  0.042049  0.039947  0.015481
```

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   -0.674852
1    1.025811
2    0.136964
3   -0.193952
4   -0.371576
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())
```

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

```
        # self.bias_mean = tfp.util.TransformedVariable(
        #     bijector=constrain_positive,
        #     initial_value=1.0,
        #     dtype=np.float64,
        #     name="bias_mean",
        # )
        self.bias_mean = tf.Variable(-1.5, 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
            + self.amp_gamma_L_mean * (x[..., 2]) ** 2
            + self.amp_lambda_mean * (x[..., 3]) ** 2
            + self.amp_f_mean * (x[..., 4]) ** 2
            + self.amp_r_mean * (x[..., 5]) ** 2
        )


quad_mean_fn().__call__(x=np.array([[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]]))  # should return 1
```

```
<tf.Tensor: shape=(1,), dtype=float64, numpy=array([2.5])>
```

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])
        )
```

```
class const_mean_fn(tf.Module):
    def __init__(self):
        super(const_mean_fn, self).__init__()
        self.bias_lin = tf.Variable(0.0, dtype=np.float64, name="bias_mean")

    def __call__(self, x):
        return self.bias_lin
```

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=4.0,
    dtype=np.float64,
    name="amplitude_champ",
)

observation_noise_variance_champ = tfp.util.TransformedVariable(
    bijector=constrain_positive,
    initial_value=1.,
    dtype=np.float64,
    name="observation_noise_variance_champ",
)
```

```
length_scales_champ = tfp.util.TransformedVariable(
    bijector=tfb.Sigmoid(
        np.float64(0.0),
        [1.0 / 4, 1.0 / 4, gamma_L_max / 4, lambda_max / 4, f_max / 4, r_max / 4],
    ),
    initial_value=[1 / 8, 1 / 8, gamma_L_max / 8, lambda_max / 8, f_max / 8, r_max / 8],
    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=const_mean_fn(),
)

print(champ_GP.trainable_variables)

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

```
(<tf.Variable 'amplitude_champ:0' shape=() dtype=float64, numpy=1.3862943611198906>, <tf.Var
```

Train the Hyperparameters

Leave One Out Predictive Log-likelihood

```
# 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)
```

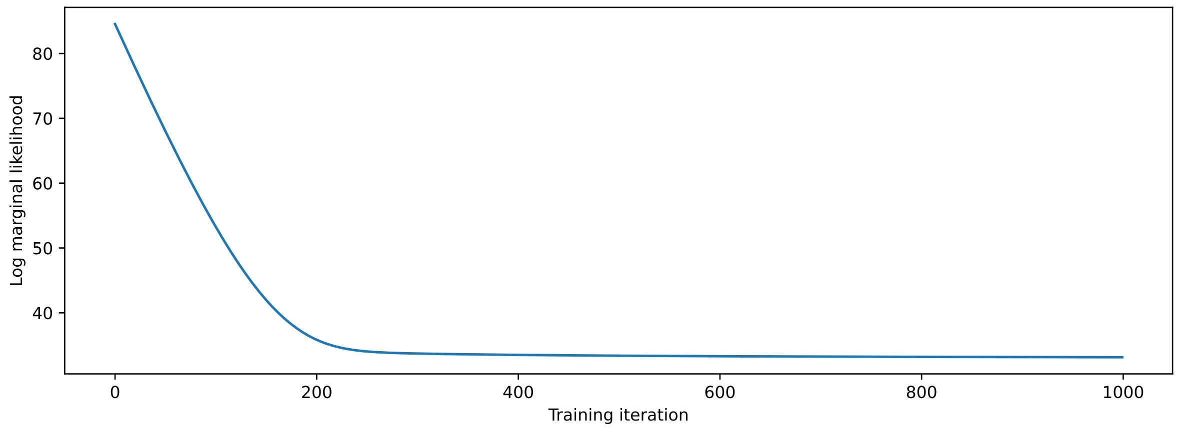
```
#         K_inv = tf.linalg.inv(K)
#         K_inv_y = K_inv @ tf.reshape(obs_vals - means, shape=[obs_vals.shape[0], 1])
#         K_inv_diag = tf.linalg.diag_part(K_inv)
#         log_var = tf.math.log(K_inv_diag)
#         log_mu = tf.reshape(K_inv_y, shape=[-1]) ** 2
#         loss = -tf.math.reduce_sum(log_var - 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
```

Maximum Likelihood Estimation

```
# Now we optimize the model parameters.
num_iters = 1000

# Use `tf.function` to trace the loss for more efficient evaluation.
@tf.function(autograph=False, jit_compile=False)
def train_model():
    with tf.GradientTape() as tape:
        loss = -champ_GP.log_prob(obs_vals)
    grads = tape.gradient(loss, champ_GP.trainable_variables)
```

```python
    Adam_optim.apply_gradients(zip(grads, champ_GP.trainable_variables))
    return loss


# Store the likelihood values during training, so we can plot the progress
lls_ = np.zeros(num_iters, np.float64)
for i in range(num_iters):
    loss = train_model()
    lls_[i] = loss

print("Trained parameters:")
print("amplitude: {}".format(amplitude_champ._value().numpy()))
print("length_scales: {}".format(length_scales_champ._value().numpy()))
print(
    "observation_noise_variance: {}".format(
        observation_noise_variance_champ._value().numpy()
    )
)

# Plot the loss evolution
plt.figure(figsize=(12, 4))
plt.plot(lls_)
plt.xlabel("Training iteration")
plt.ylabel("Log marginal likelihood")
plt.show()
```

```
Trained parameters:
amplitude: 0.6075224096984188
length_scales: [0.24927684 0.24944805 0.0083141  0.01726937 0.01781074 0.01781126]
observation_noise_variance: 0.013020897865743602
```

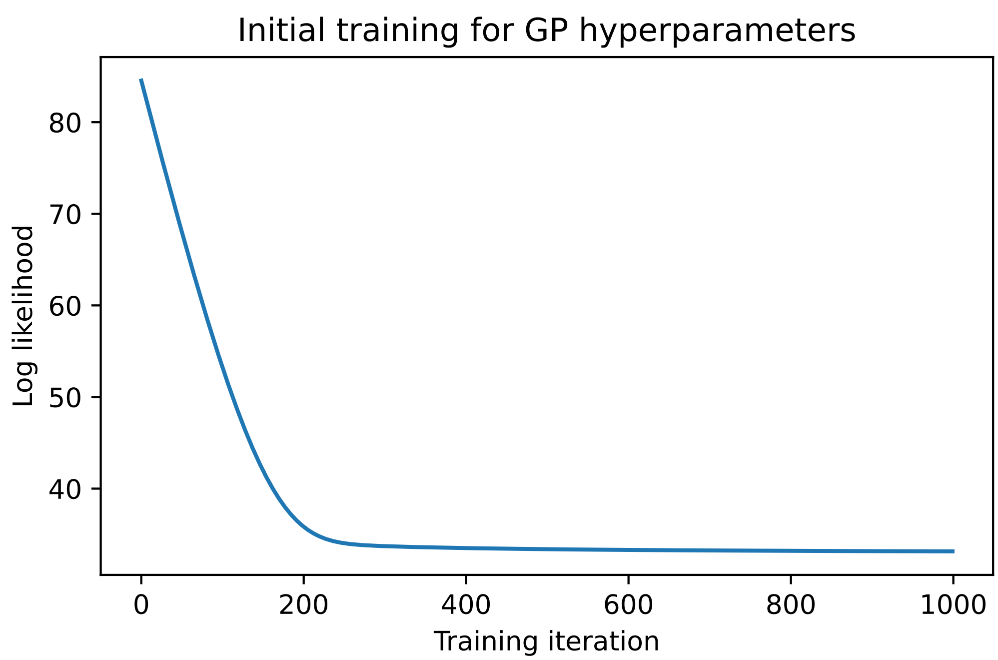



```
print("Trained parameters:")
for var in champ_GP.trainable_variables:
    if "bias" in var.name:
        print("{} is {}\n".format(var.name, var.numpy().round(3)))
    else:
        if "length" in var.name:
            print(
                "{} is {}\n".format(
                    var.name,
                    tfb.Sigmoid(
                        np.float64(0.0),
                        [
                            1.0 / 4,
                            1.0 / 4,
                            gamma_L_max / 4,
                            lambda_max / 4,
                            f_max / 4,
                            r_max / 4,
                        ],
                    )
                    .forward(var)
                    .numpy()
                    .round(3),
                )
            )
        else:
            print(
                "{} is {}\n".format(
                    var.name, constrain_positive.forward(var).numpy().round(3)
```

```
                )
            )
```

```
Trained parameters:
amplitude_champ:0 is 0.608

length_scales_champ:0 is [0.249 0.249 0.008 0.017 0.018 0.018]

observation_noise_variance_champ:0 is 0.013

bias_mean:0 is 0.122
```

```
plt.figure(figsize=(6, 3.5))
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_log_discrep.pdf")
plt.show()
```



Creating slices across one variable dimension

```
plot_samp_no = 21
plot_gp_no = 100
gp_samp_no = 30
```

```
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,
    ), 5, 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,
    ), 5, 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
            np.linspace(0, gamma_L_max, plot_samp_no, dtype=np.float64).reshape(-1, 1),  # ga
            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,
    ), 5, 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(0, gamma_L_max, plot_gp_no, dtype=np.float64).reshape(-1, 1),  # gamm
            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
            np.linspace(0, lambda_max, plot_samp_no, dtype=np.float64).reshape(-1, 1), # laml
            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,
```

```
    ), 5, 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(0, lambda_max, plot_gp_no, dtype=np.float64).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,
    ),
    "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
            np.linspace(0, f_max, plot_samp_no, dtype=np.float64).reshape(-1, 1), # f
            np.repeat(pv_champ_r, plot_samp_no).reshape(-1, 1),  # r
        ),
        axis=1,
    ), 5, 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(0, f_max, plot_gp_no, dtype=np.float64).reshape(-1, 1), # 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
            np.linspace(0, r_max, plot_samp_no, dtype=np.float64).reshape(-1, 1), # r
```

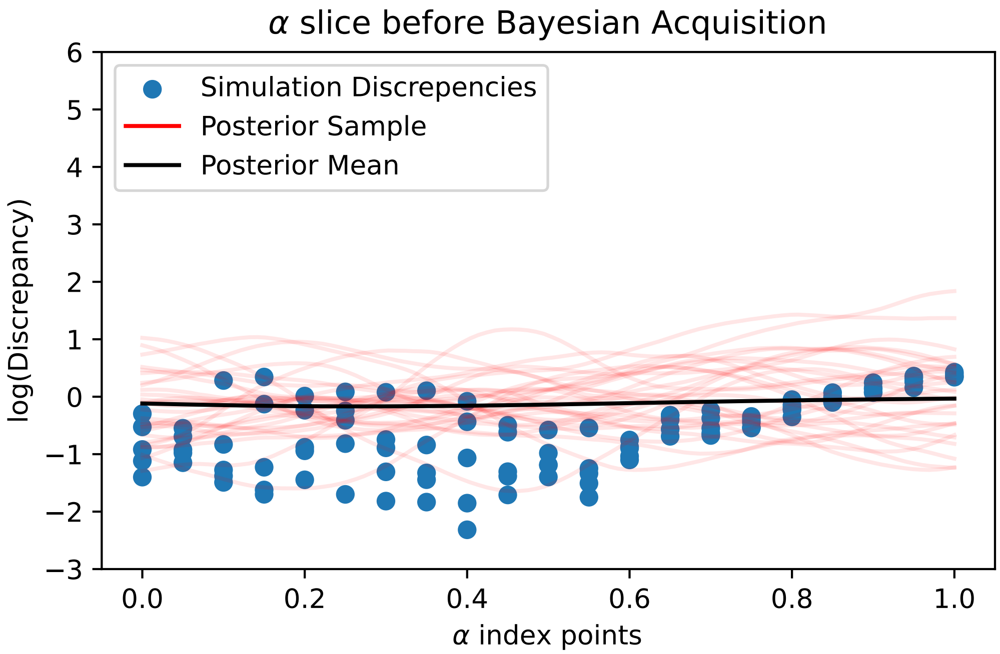
```
        ),
        axis=1,
    ), 5, axis = 0),
    "r_gp_samples": np.concatenate(
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 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(0, r_max, plot_gp_no, dtype=np.float64).reshape(-1, 1), # r
        ),
        axis=1,
    ),
}
```

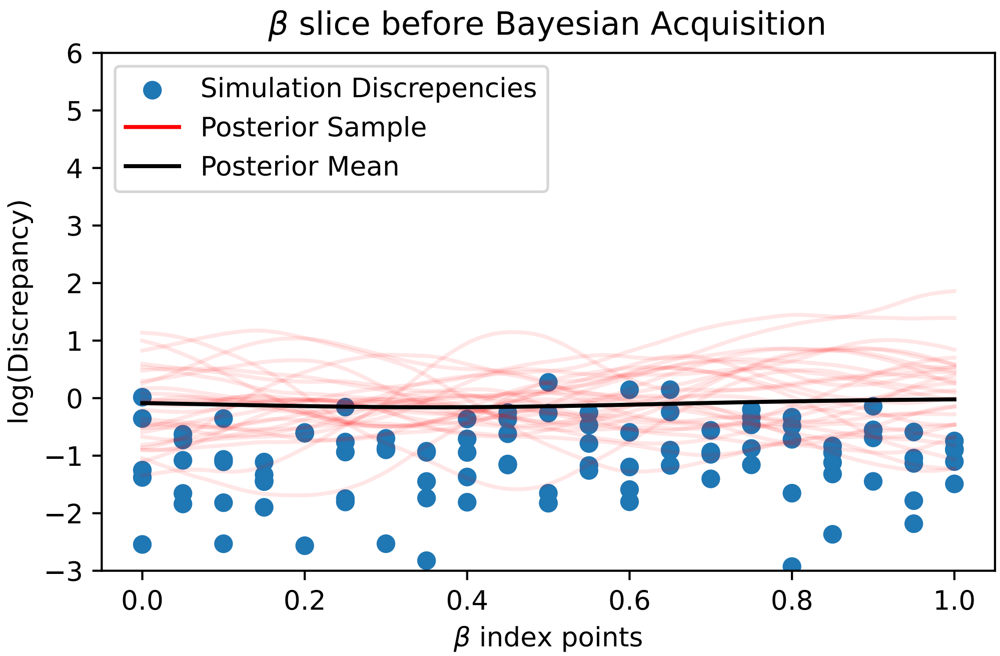
Plotting the GPs across different slices

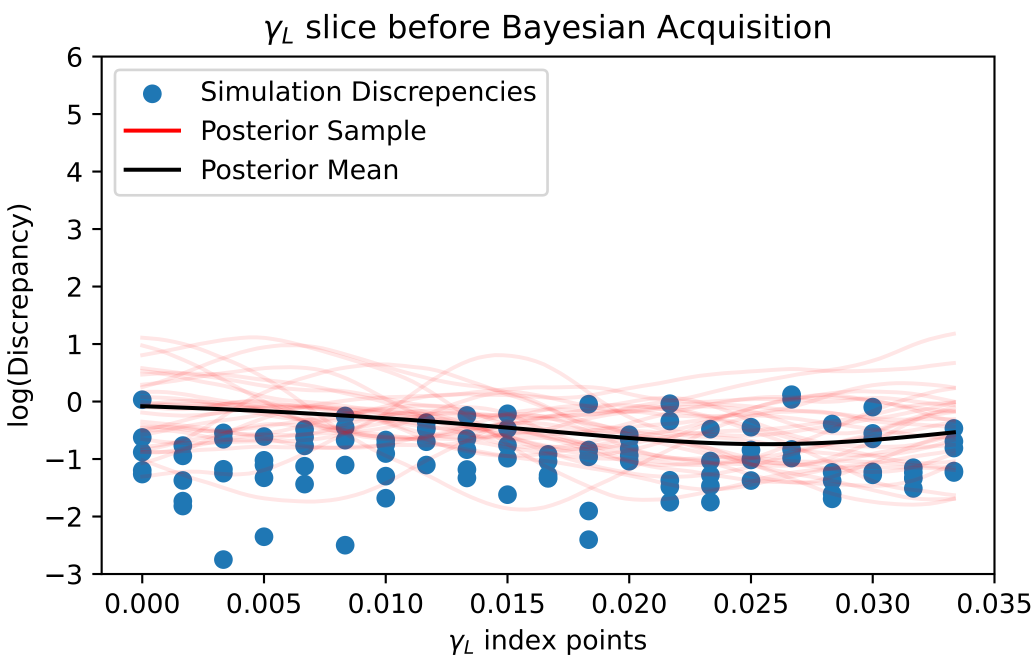
```
GP_seed = tfp.random.sanitize_seed(4362)
vars = ["alpha", "beta", "gamma_L", "lambda", "f", "r"]
slice_indices_dfs_dict = {}
slice_index_vals_dict = {}
slice_discrepencies_dict = {}

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"], mean_of = 1
        ),
        axis=1,
    )
    slice_discrepencies_dict[var + "_slice_discrepencies"] = discreps


    gp_samples_df = pd.DataFrame(
```

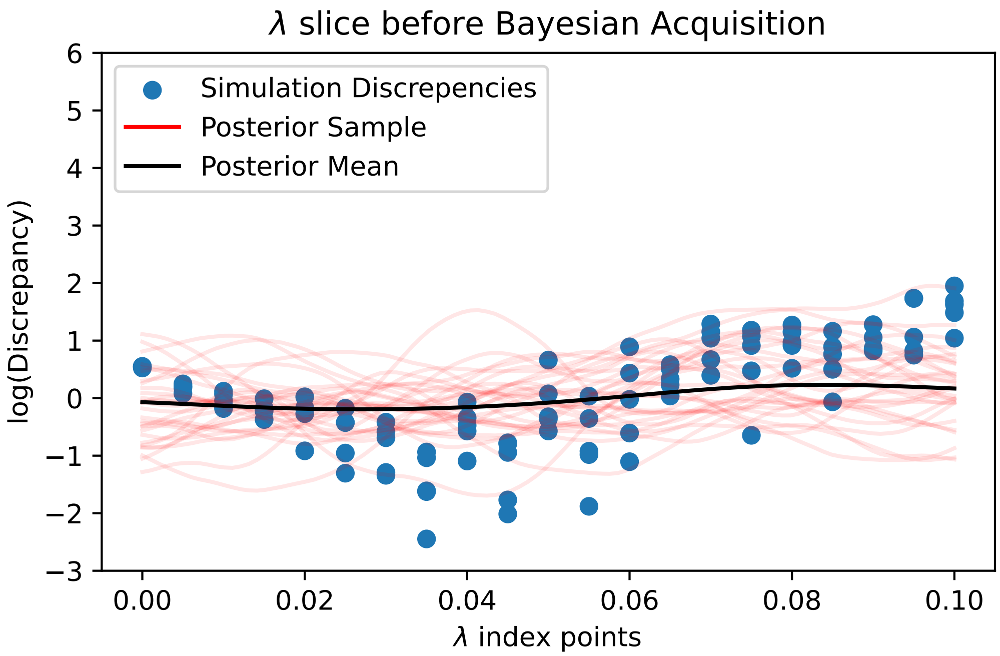
```
        slice_samples_dict[var + "_gp_samples"], columns=variables_names
    )
    slice_indices_dfs_dict[var + "_gp_indices_df"] = gp_samples_df
    slice_index_vals_dict[var + "_gp_index_vals"] = gp_samples_df.values

    champ_GP_reg_plot = 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=const_mean_fn(),
    )
    GP_samples = champ_GP_reg_plot.sample(gp_samp_no, seed=GP_seed)

    plt.figure(figsize=(6, 3.5))
    plt.scatter(
        val_df[var].values,
        discreps,
        label = "Simulation Discrepencies",
    )
    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,
        )
    plt.plot(
        slice_indices_dfs_dict[var + "_gp_indices_df"][var].values,
        champ_GP_reg_plot.mean_fn(slice_indices_dfs_dict[var + "_gp_indices_df"].values),
        c="black",
        alpha=1,
        label="Posterior Mean",
    )
    leg = plt.legend(loc="upper left")
    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("log(Discrepancy)")
    plt.ylim((-3, 6))
    plt.savefig("champagne_GP_images/initial_" + var + "_slice_log_discrep.pdf")
    plt.show()
```

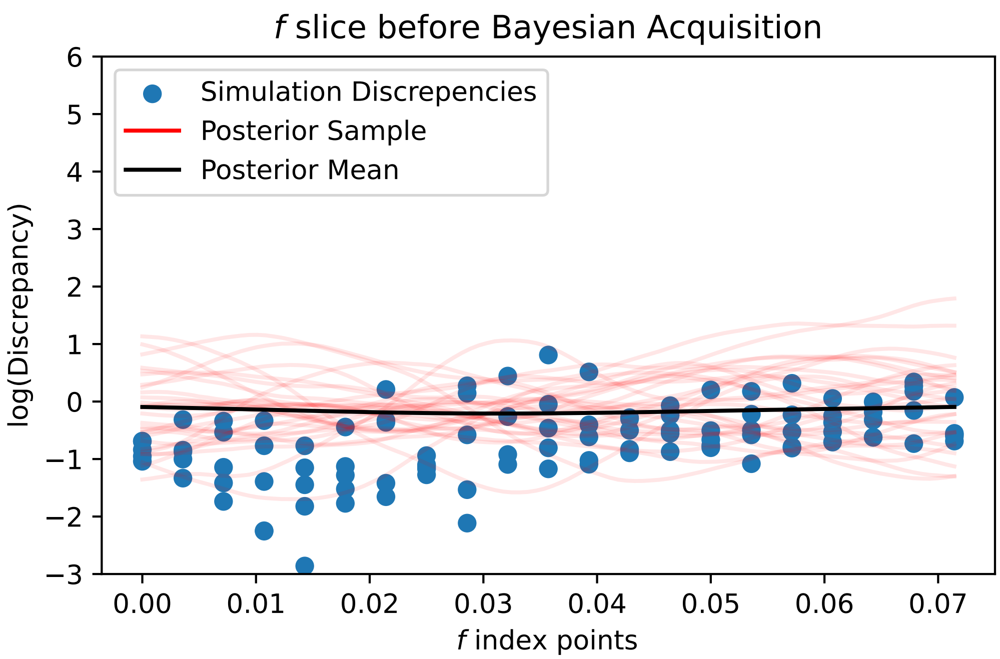



β slice before Bayesian Acquisition
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
β index points
6
5
4
3
2
1
0
−1
−2
−3
0.0
0.2
0.4
0.6
0.8
1.0

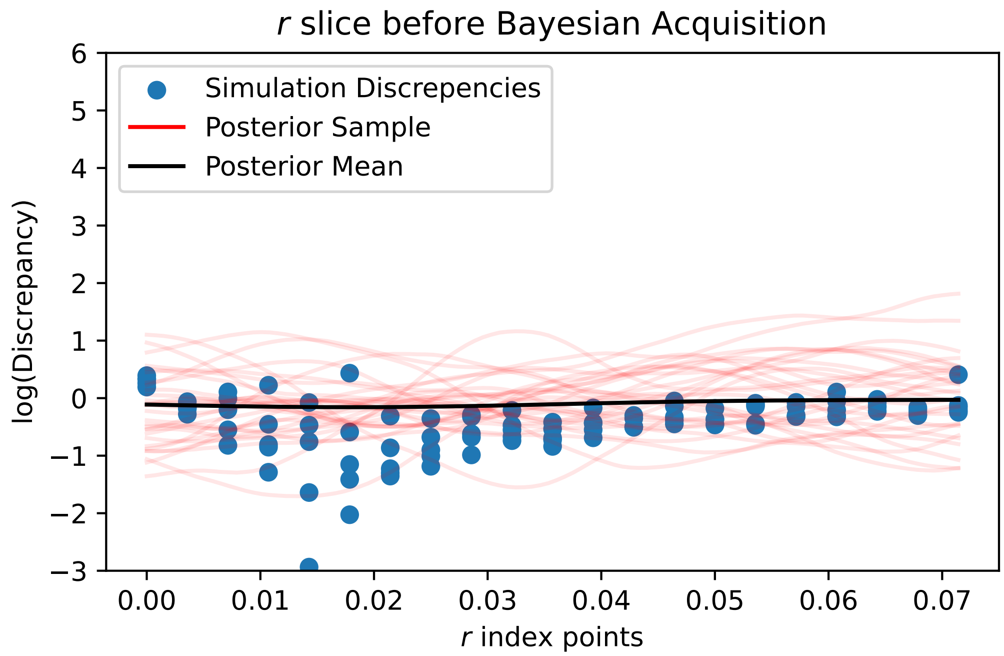

γ_L slice before Bayesian Acquisition
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
γ_L index points
6
5
4
3
2
1
0
−1
−2
−3
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035



λ slice before Bayesian Acquisition
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
λ index points



f slice before Bayesian Acquisition
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
f index points



Acquiring the next datapoint to test

Proof that .variance returns what we need in acquisition function

```
champ_GP_reg = tfd.GaussianProcessRegressionModel(
    kernel=kernel_champ,
    observation_index_points=index_vals,
    observations=obs_vals,
    observation_noise_variance=observation_noise_variance_champ,
    mean_fn=const_mean_fn(),
)

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 0.369
Others Kernel is [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[ 2.67031081e+00  1.03204138e-03 -7.48102384e-02 ...  1.65960671e-06
  -7.75717474e-02  1.24798171e-03]
 [ 1.03204138e-03  2.62445135e+00  1.22438395e-04 ...  1.26371308e-03
  -1.94464820e-02 -2.36175457e-04]
 [-7.48102384e-02  1.22438395e-04  2.62884796e+00 ...  1.35385431e-03
  -8.32562719e-04 -2.21762817e-03]
 ...
 [ 1.65960671e-06  1.26371308e-03  1.35385431e-03 ...  2.76403097e+00
  -1.41456401e-03 -1.65986033e-04]
 [-7.75717474e-02 -1.94464820e-02 -8.32562719e-04 ... -1.41456401e-03
   2.84877241e+00  1.21819757e-02]
 [ 1.24798171e-03 -2.36175457e-04 -2.21762817e-03 ... -1.65986033e-04
   1.21819757e-02  2.66725867e+00]]
Variance function is [0.382]
Variance function is 0.369
```

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=gamma_L_max/2,
    bijector=tfb.Sigmoid(np.float64(0.), gamma_L_max),
    dtype=np.float64,
    name="next_gamma_L",
)

next_lambda = tfp.util.TransformedVariable(
    initial_value=lambda_max/2,
    bijector=tfb.Sigmoid(np.float64(0.), lambda_max),
    dtype=np.float64,
    name="next_lambda",
)

next_f = tfp.util.TransformedVariable(
    initial_value=f_max/2,
    bijector=tfb.Sigmoid(np.float64(0.), f_max),
    dtype=np.float64,
    name="next_f",
)

next_r = tfp.util.TransformedVariable(
    initial_value=r_max/2,
    bijector=tfb.Sigmoid(np.float64(0.), r_max),
    dtype=np.float64,
    name="next_r",
)

next_vars = (
    (next_alpha.trainable_variables[0],
    next_beta.trainable_variables[0],
    next_gamma_L.trainable_variables[0],
    next_lambda.trainable_variables[0],
    next_f.trainable_variables[0],
    next_r.trainable_variables[0],)
)
```

```
next_vars
```

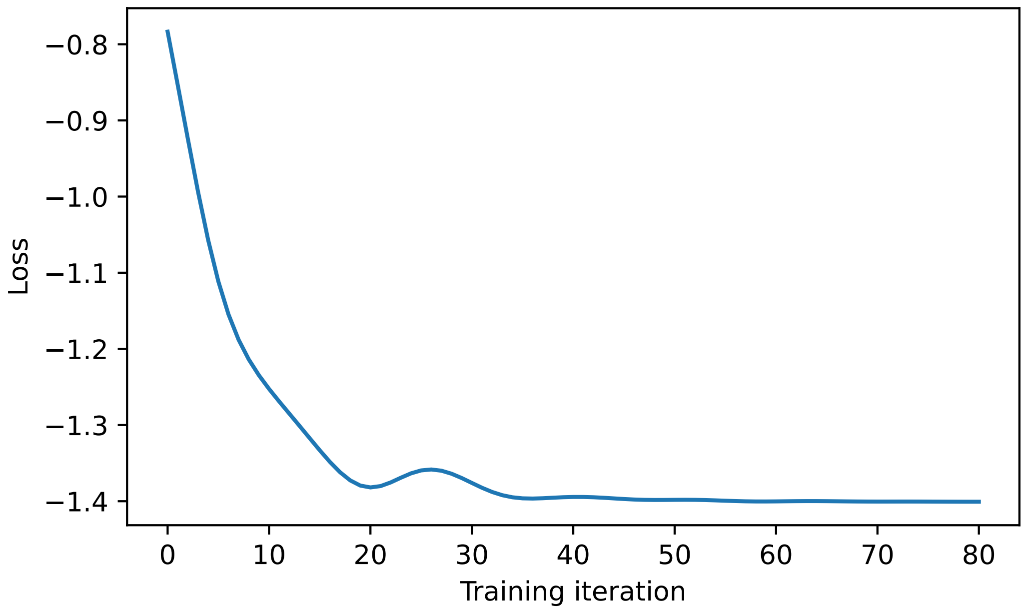
```
(<tf.Variable 'next_alpha:0' shape=() dtype=float64, numpy=0.0>,
 <tf.Variable 'next_beta:0' shape=() dtype=float64, numpy=0.0>,
 <tf.Variable 'next_gamma_L:0' shape=() dtype=float64, numpy=0.0>,
 <tf.Variable 'next_lambda:0' shape=() dtype=float64, numpy=0.0>,
 <tf.Variable 'next_f:0' shape=() dtype=float64, numpy=0.0>,
 <tf.Variable 'next_r:0' shape=() dtype=float64, numpy=0.0>)
```

```
eta_t = tf.constant(1.0, dtype=np.float64)

def UCB_loss(champ_GP_reg):
    next_guess = tf.reshape(
        tf.stack([next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]),
        [1, 6],
    )
    mean_t = champ_GP_reg.mean_fn(next_guess)
    std_t = tf.math.sqrt(
        champ_GP_reg.variance(index_points=next_guess)
        - observation_noise_variance_champ
    )
    return tf.squeeze(mean_t - std_t)


optimizer_fast = tf.keras.optimizers.Adam(learning_rate=0.1)

@tf.function(autograph=False, jit_compile=False)
def opt_var():
    with tf.GradientTape() as tape:
        loss = UCB_loss(champ_GP_reg)
    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("Trained parameters:")
for var in [next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]:
    print("{} is {}".format(var.name, (var.bijector.forward(var).numpy().round(3))))
```

```
Acquisition function convergence reached at iteration 81.
Trained parameters:
next_alpha is 0.639
next_beta is 0.565
next_gamma_L is 0.017
next_lambda is 0.051
next_f is 0.036
next_r is 0.036
```

```
plt.figure(figsize=(6, 3.5))
plt.plot(lls_)
plt.xlabel("Training iteration")
plt.ylabel("Loss")
plt.savefig("champagne_GP_images/bolfi_optim_loss_log_discrep.pdf")
plt.show()
```




```
def update_GP_LOO():
    @tf.function(autograph=False, jit_compile=False)
    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()

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

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


def update_GP_MLE(champ_GP):
    @tf.function(autograph=False, jit_compile=False)
    def train_model():
        with tf.GradientTape() as tape:
            loss = -champ_GP.log_prob(obs_vals)
        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 = train_model()

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

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

```python
# def UCB_loss(eta_t, champ_GP_reg):
#     next_guess = tf.reshape(
#         tf.stack([next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]),
#         [1, 6],
#     )
#     mean_t = champ_GP_reg.mean_fn(next_guess)
#     std_t = champ_GP_reg.stddev(index_points=next_guess)
#     return tf.squeeze(mean_t - eta_t * std_t)


def update_var_UCB(eta_t, champ_GP_reg, next_vars):
    optimizer_fast = tf.keras.optimizers.Adam(learning_rate=0.1)

    @tf.function(autograph=False, jit_compile=False)
    def opt_var():
        with tf.GradientTape() as tape:
            loss = UCB_loss(eta_t, champ_GP_reg)
        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-3  # 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}.")
            break

        previous_loss = loss

    next_guess = tf.reshape(
        tf.stack([next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]),
        [1, 6],
    )
```

```python
    print(
        "The final UCB loss was {}".format(loss.numpy().round(3))
        + " with predicted mean of {}".format(
            champ_GP_reg.mean_fn(next_guess).numpy().round(3)
        )
    )
    for var in optimizer_fast.variables:
        var.assign(tf.zeros_like(var))


def update_var_EI(GP_reg, alpha, beta, gamma_L, lambda_, f, r, min_obs):
    def EI_loss(alpha, beta, gamma_L, lambda_, f, r, min_obs):
        next_guess = tf.reshape(
            tf.stack([alpha, beta, gamma_L, lambda_, f, r]),
            [1, 6],
        )
        mean_t = GP_reg.mean_fn(next_guess)
        std_t = GP_reg.stddev(index_points=next_guess)
        delt = min_obs - mean_t
        return -tf.squeeze(
            delt * tfd.Normal(0, std_t).cdf(delt)
            + std_t * GP_reg.prob(delt, index_points=next_guess)
        )

    optimizer_fast = tf.keras.optimizers.Adam(learning_rate=0.1)

    @tf.function(autograph=False, jit_compile=False)
    def opt_var():
        with tf.GradientTape() as tape:
            loss = EI_loss(alpha, beta, gamma_L, lambda_, f, r, min_obs)
        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-9  # Set your desired tolerance level
    previous_loss = np.float64("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

    next_guess = tf.reshape(
        tf.stack([next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]),
        [1, 6],
    )
    print(
        "The final EI loss was {}".format(loss.numpy().round(3))
        + " with predicted mean of {}".format(
            champ_GP_reg.mean_fn(next_guess).numpy().round(3)
        )
    )


# update_var_EI(
#     champ_GP_reg, next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r
# )
# 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))
    return np.sqrt(np.log((t + 1) ** (d * 2 + 2) * np.pi**2 / (3 * exploration_rate)))
```

```
# optimizer_fast = tf.keras.optimizers.Adam(learning_rate=1.)
# update_var_EI()
# plt.figure(figsize=(6, 3.5))
# plt.plot(lls_)
# plt.xlabel("Training iteration")
# plt.ylabel("Loss")
# plt.show()
```

```
num_slice_updates = 11

all_slices = [np.linspace(0, 1, num_slice_updates, dtype=np.float64),  # alpha
        np.linspace(0, 1, num_slice_updates, dtype=np.float64),  # beta
        np.linspace(0, gamma_L_max, num_slice_updates, dtype=np.float64),  # gamma_L
        np.linspace(0, lambda_max, num_slice_updates, dtype=np.float64),  # lambda
        np.linspace(0, f_max, num_slice_updates, dtype=np.float64),  # f
        np.linspace(0, r_max, num_slice_updates, dtype=np.float64),  # r
]
```

```
exploration_rate = 1
d = 6
update_GP_hp_freq = 20  # how many iterations before updating GP hyperparams
eta_t = tf.Variable(0, dtype=np.float64, name="eta_t")
min_obs = tf.Variable(100, dtype=np.float64, name="min_obs", shape=())
min_index = index_vals[
    champ_GP_reg.mean_fn(index_vals) == min(champ_GP_reg.mean_fn(index_vals))
][0]
simulation_reps = 20

for t in range(201):
    min_index = index_vals[
        champ_GP_reg.mean_fn(index_vals) == min(champ_GP_reg.mean_fn(index_vals))
    ][
        0,
    ]
    optimizer_slow = tf.keras.optimizers.Adam()
    eta_t.assign(new_eta_t(t, d, exploration_rate))
    min_obs.assign(min(champ_GP_reg.mean_fn(index_vals)))
    print("Iteration " + str(t))
    # print(eta_t)

    ######################################################################

    # for var in [next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]:
    #     var.assign(
    #         var.bijector.forward(np.float64(100000000.0))
    #         * np.float64(np.random.uniform())
    #     )

    index_update = 0
    for var in [next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r]:
```

```
        if np.random.uniform() > 0.2:
            var.assign(min_index[index_update])
        else:
            var.assign(
                var.bijector.forward(np.float64(100000000.0))
                * np.float64(np.random.uniform())
            )
        index_update += 1

    # update_var_UCB(eta_t, champ_GP_reg)
    update_var_EI(
        champ_GP_reg,
        next_alpha,
        next_beta,
        next_gamma_L,
        next_lambda,
        next_f,
        next_r,
        min_obs,
    )

    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("The next parameters to simulate from are {}".format(new_params.round(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)

    print("The mean of the samples was {}".format(new_discrepency.round(3)))

    slice_var = [next_alpha, next_beta, next_gamma_L, next_lambda, next_f, next_r][t % 6]
    for val in all_slices[t % 6]:
        if np.random.uniform() < 1/5 + np.exp(1 - t/4):
            slice_var.assign(val)

            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)

            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)

    ######################################################################

    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=const_mean_fn(),
```

```
    )

    if t % update_GP_hp_freq == 0:
        champ_GP = tfd.GaussianProcess(
            kernel=kernel_champ,
            observation_noise_variance=observation_noise_variance_champ,
            index_points=index_vals,
            mean_fn=const_mean_fn(),
        )
        # update_GP_LOO()
        update_GP_MLE(champ_GP)
        min_value = min(champ_GP_reg.mean_fn(index_vals))
        min_index = index_vals[champ_GP_reg.mean_fn(index_vals) == min_value][0,]
        print(
            "The minimum predicted mean of the observed indices is {}".format(
                min_value.numpy().round(3)
            )
            + " at the point \n{}".format(min_index.round(3))
        )

    if (t > 0) & (t % 50 == 0):
        print("Trained parameters:")
        for train_var in champ_GP.trainable_variables:
            if "bias" in train_var.name:
                print("{} is {}\n".format(train_var.name, train_var.numpy().round(3)))
            else:
                if "length" in train_var.name:
                    print(
                        "{} is {}\n".format(
                            train_var.name,
                            tfb.Sigmoid(
                                np.float64(0.0),
                                [
                                    1.0 / 4,
                                    1.0 / 4,
                                    gamma_L_max / 4,
                                    lambda_max / 4,
                                    f_max / 4,
                                    r_max / 4,
                                ],
                            )
                            .forward(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_plot = 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=const_mean_fn(),
    )
    GP_samples = champ_GP_reg_plot.sample(gp_samp_no, seed=GP_seed)

    plt.figure(figsize=(6, 3.5))
    plt.scatter(
        slice_indices_dfs_dict[var + "_slice_indices_df"][var].values,
        slice_discrepencies_dict[var + "_slice_discrepencies"],
        label="Simulation Discrepencies",
    )
    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,
        )
    plt.plot(
        slice_indices_dfs_dict[var + "_gp_indices_df"][var].values,
        champ_GP_reg_plot.mean_fn(
            slice_indices_dfs_dict[var + "_gp_indices_df"].values
```

```
                ),
                c="black",
                alpha=1,
                label="Posterior Mean",
            )
            leg = plt.legend(loc="upper left")
            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("log(Discrepancy)")
            plt.ylim((-3, 6))
            plt.savefig(
                "champagne_GP_images/"
                + var
                + "_slice_"
                + str(t)
                + "_bolfi_updates_log_discrep.pdf"
            )
            plt.show()
```

```
Iteration 0
Acquisition function convergence reached at iteration 486.
The final EI loss was -0.36 with predicted mean of [-0.503]
The next parameters to simulate from are [[0.596 0.378 0.026 0.058 0.021 0.034]]
The mean of the samples was -0.356
Hyperparameter convergence reached at iteration 5086.
The minimum predicted mean of the observed indices is -1.085 at the point
[0.69  0.206 0.029 0.053 0.02  0.016]
Iteration 1
Acquisition function convergence reached at iteration 171.
The final EI loss was -0.399 with predicted mean of [-0.543]
The next parameters to simulate from are [[0.453 0.515 0.01  0.02  0.035 0.033]]
The mean of the samples was -0.519
```

```
Iteration 2
Acquisition function convergence reached at iteration 670.
The final EI loss was -0.399 with predicted mean of [-0.543]
The next parameters to simulate from are [[0.423 0.419 0.01  0.022 0.036 0.034]]
The mean of the samples was -0.603
Iteration 3
Acquisition function convergence reached at iteration 120.
The final EI loss was -0.013 with predicted mean of [-1.108]
The next parameters to simulate from are [[0.669 0.211 0.029 0.05  0.019 0.016]]
The mean of the samples was -1.136
Iteration 4
Acquisition function convergence reached at iteration 5.
The final EI loss was 0.0 with predicted mean of [0.781]
The next parameters to simulate from are [[0.647 0.218 0.025 0.098 0.021 0.016]]
The mean of the samples was 0.946
Iteration 5
Acquisition function convergence reached at iteration 131.
The final EI loss was -0.019 with predicted mean of [-1.145]
The next parameters to simulate from are [[0.68  0.211 0.029 0.044 0.02  0.016]]
The mean of the samples was -1.233
Iteration 6
The final EI loss was -0.398 with predicted mean of [-0.58]
The next parameters to simulate from are [[0.65  0.22  0.029 0.026 0.018 0.019]]
The mean of the samples was -0.589
Iteration 7
Acquisition function convergence reached at iteration 279.
The final EI loss was -0.399 with predicted mean of [-0.577]
The next parameters to simulate from are [[0.103 0.15  0.023 0.028 0.01  0.019]]
The mean of the samples was -0.618
Iteration 8
Acquisition function convergence reached at iteration 144.
The final EI loss was -0.028 with predicted mean of [-1.19]
The next parameters to simulate from are [[0.671 0.211 0.029 0.044 0.02  0.019]]
The mean of the samples was -1.084
Iteration 9
Acquisition function convergence reached at iteration 138.
The final EI loss was -0.008 with predicted mean of [-1.234]
The next parameters to simulate from are [[0.68  0.211 0.031 0.044 0.02  0.019]]
The mean of the samples was -1.246
Iteration 10
Acquisition function convergence reached at iteration 163.
The final EI loss was -0.399 with predicted mean of [-0.617]
The next parameters to simulate from are [[0.149 0.233 0.023 0.029 0.009 0.017]]
```

```
The mean of the samples was -0.544
Iteration 11
Acquisition function convergence reached at iteration 111.
The final EI loss was -0.394 with predicted mean of [-0.61]
The next parameters to simulate from are [[0.662 0.208 0.003 0.043 0.021 0.018]]
The mean of the samples was -0.744
Iteration 12
Acquisition function convergence reached at iteration 145.
The final EI loss was -0.007 with predicted mean of [-1.245]
The next parameters to simulate from are [[0.679 0.212 0.031 0.043 0.02  0.019]]
The mean of the samples was -1.152
Iteration 13
Acquisition function convergence reached at iteration 504.
The final EI loss was -0.399 with predicted mean of [-0.64]
The next parameters to simulate from are [[0.572 0.216 0.031 0.025 0.019 0.019]]
The mean of the samples was -0.508
Iteration 14
Acquisition function convergence reached at iteration 4606.
The final EI loss was -0.399 with predicted mean of [-0.64]
The next parameters to simulate from are [[0.417 0.282 0.011 0.018 0.037 0.033]]
The mean of the samples was -0.586
Iteration 15
Acquisition function convergence reached at iteration 134.
The final EI loss was -0.399 with predicted mean of [-0.64]
The next parameters to simulate from are [[0.133 0.153 0.023 0.031 0.006 0.019]]
The mean of the samples was -0.648
Iteration 16
The final EI loss was -0.395 with predicted mean of [-0.645]
The next parameters to simulate from are [[0.658 0.182 0.033 0.057 0.02  0.014]]
The mean of the samples was -0.585
Iteration 17
Acquisition function convergence reached at iteration 103.
The final EI loss was 0.0 with predicted mean of [1.262]
The next parameters to simulate from are [[0.256 0.333 0.022 0.097 0.012 0.015]]
The mean of the samples was 1.586
Iteration 18
Acquisition function convergence reached at iteration 167.
The final EI loss was -0.399 with predicted mean of [-0.64]
The next parameters to simulate from are [[0.626 0.224 0.002 0.042 0.02  0.021]]
The mean of the samples was -0.66
Iteration 19
Acquisition function convergence reached at iteration 9443.
The final EI loss was -0.399 with predicted mean of [-0.639]
```

```
The next parameters to simulate from are [[0.792 0.212 0.03  0.035 0.02  0.019]]
The mean of the samples was -0.597
Iteration 20
Acquisition function convergence reached at iteration 5293.
The final EI loss was -0.399 with predicted mean of [-0.64]
The next parameters to simulate from are [[0.158 0.161 0.022 0.029 0.003 0.019]]
The mean of the samples was -0.699
Hyperparameter convergence reached at iteration 2783.
The minimum predicted mean of the observed indices is -1.271 at the point
[0.6   0.212 0.031 0.043 0.02  0.019]
Iteration 21
Acquisition function convergence reached at iteration 1995.
The final EI loss was -0.399 with predicted mean of [-0.635]
The next parameters to simulate from are [[0.431 0.158 0.031 0.038 0.02  0.02 ]]
The mean of the samples was -0.855
Iteration 22
Acquisition function convergence reached at iteration 8556.
The final EI loss was -0.399 with predicted mean of [-0.638]
The next parameters to simulate from are [[0.658 0.157 0.033 0.055 0.04  0.015]]
The mean of the samples was -0.407
Iteration 23
Acquisition function convergence reached at iteration 131.
The final EI loss was -0.01 with predicted mean of [-1.293]
The next parameters to simulate from are [[0.618 0.223 0.031 0.042 0.02  0.019]]
The mean of the samples was -1.366
Iteration 24
Acquisition function convergence reached at iteration 2799.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.221 0.209 0.027 0.027 0.013 0.022]]
The mean of the samples was -0.727
Iteration 25
Acquisition function convergence reached at iteration 7369.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.136 0.212 0.026 0.025 0.017 0.023]]
The mean of the samples was -0.653
Iteration 26
Acquisition function convergence reached at iteration 2692.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.644 0.173 0.033 0.052 0.004 0.017]]
The mean of the samples was -1.37
Iteration 27
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.499]
```

```
The next parameters to simulate from are [[0.618 0.223 0.031 0.085 0.02  0.019]]
The mean of the samples was 0.44
Iteration 28
Acquisition function convergence reached at iteration 150.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.651 0.194 0.031 0.053 0.037 0.016]]
The mean of the samples was -0.529
Iteration 29
Acquisition function convergence reached at iteration 172.
The final EI loss was -0.318 with predicted mean of [-1.664]
The next parameters to simulate from are [[0.61  0.203 0.032 0.046 0.002 0.02 ]]
The mean of the samples was -1.057
Iteration 30
Acquisition function convergence reached at iteration 681.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.62  0.199 0.033 0.041 0.02  0.04 ]]
The mean of the samples was -0.731
Iteration 31
Acquisition function convergence reached at iteration 347.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.402 0.31  0.027 0.022 0.033 0.031]]
The mean of the samples was -0.681
Iteration 32
Acquisition function convergence reached at iteration 418.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.134 0.179 0.029 0.027 0.014 0.023]]
The mean of the samples was -0.589
Iteration 33
Acquisition function convergence reached at iteration 135.
The final EI loss was -0.004 with predicted mean of [-1.335]
The next parameters to simulate from are [[0.612 0.222 0.031 0.043 0.019 0.02 ]]
The mean of the samples was -1.226
Iteration 34
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.314]
The next parameters to simulate from are [[0.727 0.223 0.031 0.005 0.02  0.019]]
The mean of the samples was 0.236
Iteration 35
Acquisition function convergence reached at iteration 3642.
The final EI loss was -0.399 with predicted mean of [-0.648]
The next parameters to simulate from are [[0.674 0.214 0.025 0.047 0.02  0.01 ]]
The mean of the samples was -0.733
Iteration 36
```

```
Acquisition function convergence reached at iteration 159.
The final EI loss was -0.399 with predicted mean of [-0.648]
The next parameters to simulate from are [[0.458 0.508 0.015 0.02  0.035 0.034]]
The mean of the samples was -0.429
Iteration 37
Acquisition function convergence reached at iteration 779.
The final EI loss was -0.399 with predicted mean of [-0.648]
The next parameters to simulate from are [[0.75  0.212 0.028 0.048 0.022 0.027]]
The mean of the samples was -0.703
Iteration 38
Acquisition function convergence reached at iteration 3217.
The final EI loss was -0.399 with predicted mean of [-0.648]
The next parameters to simulate from are [[0.825 0.237 0.03  0.044 0.02  0.019]]
The mean of the samples was -0.667
Iteration 39
Acquisition function convergence reached at iteration 641.
The final EI loss was -0.399 with predicted mean of [-0.648]
The next parameters to simulate from are [[0.457 0.355 0.03  0.021 0.035 0.031]]
The mean of the samples was -0.65
Iteration 40
The final EI loss was -0.396 with predicted mean of [-0.643]
The next parameters to simulate from are [[0.565 0.237 0.03  0.036 0.004 0.021]]
The mean of the samples was -0.809
Hyperparameter convergence reached at iteration 1855.
The minimum predicted mean of the observed indices is -1.379 at the point
[0.565 0.237 0.03  0.036 0.05  0.021]
Iteration 41
Acquisition function convergence reached at iteration 120.
The final EI loss was -0.007 with predicted mean of [-1.392]
The next parameters to simulate from are [[0.551 0.243 0.029 0.035 0.05  0.022]]
The mean of the samples was -1.117
Iteration 42
Acquisition function convergence reached at iteration 114.
The final EI loss was -0.005 with predicted mean of [-1.302]
The next parameters to simulate from are [[0.614 0.218 0.031 0.041 0.021 0.019]]
The mean of the samples was -1.35
Iteration 43
Acquisition function convergence reached at iteration 103.
The final EI loss was -0.003 with predicted mean of [-1.325]
The next parameters to simulate from are [[0.612 0.223 0.031 0.04  0.022 0.02 ]]
The mean of the samples was -1.546
Iteration 44
Acquisition function convergence reached at iteration 110.
```

```
The final EI loss was -0.399 with predicted mean of [-0.712]
The next parameters to simulate from are [[0.147 0.609 0.024 0.028 0.013 0.021]]
The mean of the samples was -0.654
Iteration 45
Acquisition function convergence reached at iteration 1360.
The final EI loss was -0.399 with predicted mean of [-0.711]
The next parameters to simulate from are [[0.273 0.231 0.002 0.04  0.023 0.021]]
The mean of the samples was -0.604
Iteration 46
Acquisition function convergence reached at iteration 124.
The final EI loss was -0.039 with predicted mean of [-1.479]
The next parameters to simulate from are [[0.602 0.217 0.031 0.039 0.025 0.021]]
The mean of the samples was -1.233
Iteration 47
Acquisition function convergence reached at iteration 2831.
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.767 0.44  0.032 0.036 0.016 0.018]]
The mean of the samples was -0.63
Iteration 48
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.714 0.032 0.027 0.046 0.021 0.026]]
The mean of the samples was -0.74
Iteration 49
Acquisition function convergence reached at iteration 323.
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.26  0.286 0.026 0.029 0.012 0.018]]
The mean of the samples was -0.582
Iteration 50
Acquisition function convergence reached at iteration 16.
The final EI loss was 0.0 with predicted mean of [1.143]
The next parameters to simulate from are [[0.117 0.447 0.027 0.093 0.056 0.035]]
The mean of the samples was 1.615
Trained parameters:
amplitude_champ:0 is 0.443

length_scales_champ:0 is [0.25  0.25  0.008 0.017 0.018 0.018]

observation_noise_variance_champ:0 is 0.003

bias_mean:0 is 0.173

Iteration 51
Acquisition function convergence reached at iteration 1274.
```

```
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.697 0.439 0.032 0.035 0.015 0.015]]
The mean of the samples was -1.045
Iteration 52
Acquisition function convergence reached at iteration 6408.
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.793 0.09  0.027 0.048 0.021 0.021]]
The mean of the samples was -0.693
Iteration 53
Acquisition function convergence reached at iteration 151.
The final EI loss was -0.398 with predicted mean of [-0.684]
The next parameters to simulate from are [[0.564 0.251 0.029 0.034 0.052 0.051]]
The mean of the samples was -0.673
Iteration 54
Acquisition function convergence reached at iteration 169.
The final EI loss was -0.399 with predicted mean of [-0.683]
The next parameters to simulate from are [[0.38  0.315 0.024 0.022 0.039 0.035]]
The mean of the samples was -0.799
Iteration 55
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.226]
The next parameters to simulate from are [[0.602 0.217 0.031 0.001 0.043 0.021]]
The mean of the samples was 0.264
Iteration 56
Acquisition function convergence reached at iteration 136.
The final EI loss was -0.064 with predicted mean of [-1.447]
The next parameters to simulate from are [[0.603 0.221 0.031 0.037 0.047 0.023]]
The mean of the samples was -1.391
Iteration 57
Acquisition function convergence reached at iteration 502.
The final EI loss was -0.399 with predicted mean of [-0.702]
The next parameters to simulate from are [[0.188 0.835 0.026 0.026 0.02  0.023]]
The mean of the samples was -0.765
Iteration 58
Acquisition function convergence reached at iteration 111.
The final EI loss was -0.001 with predicted mean of [-1.407]
The next parameters to simulate from are [[0.601 0.221 0.031 0.038 0.047 0.022]]
The mean of the samples was -1.293
Iteration 59
Acquisition function convergence reached at iteration 304.
The final EI loss was -0.399 with predicted mean of [-0.696]
The next parameters to simulate from are [[0.543 0.241 0.031 0.033 0.05  0.051]]
The mean of the samples was -0.751
```

```
Iteration 60
Acquisition function convergence reached at iteration 2256.
The final EI loss was -0.399 with predicted mean of [-0.696]
The next parameters to simulate from are [[0.226 0.304 0.027 0.031 0.018 0.021]]
The mean of the samples was -0.829
Hyperparameter convergence reached at iteration 569.
The minimum predicted mean of the observed indices is -1.39 at the point
[0.601 0.221 0.031 0.038 0.047 0.022]
Iteration 61
Acquisition function convergence reached at iteration 103.
The final EI loss was -0.004 with predicted mean of [-1.397]
The next parameters to simulate from are [[0.596 0.223 0.031 0.037 0.045 0.023]]
The mean of the samples was -1.128
Iteration 62
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.656]
The next parameters to simulate from are [[0.596 0.1   0.031 0.093 0.045 0.023]]
The mean of the samples was 0.913
Iteration 63
Acquisition function convergence reached at iteration 700.
The final EI loss was -0.399 with predicted mean of [-0.689]
The next parameters to simulate from are [[0.327 0.26  0.003 0.039 0.024 0.021]]
The mean of the samples was -0.94
Iteration 64
Acquisition function convergence reached at iteration 128.
The final EI loss was -0.399 with predicted mean of [-0.689]
The next parameters to simulate from are [[0.597 0.16  0.005 0.037 0.047 0.023]]
The mean of the samples was -0.912
Iteration 65
Acquisition function convergence reached at iteration 120.
The final EI loss was -0.048 with predicted mean of [-1.442]
The next parameters to simulate from are [[0.607 0.157 0.03  0.038 0.049 0.021]]
The mean of the samples was -0.971
Iteration 66
Acquisition function convergence reached at iteration 2572.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.041 0.211 0.002 0.034 0.017 0.022]]
The mean of the samples was -0.85
Iteration 67
Acquisition function convergence reached at iteration 558.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.813 0.199 0.031 0.047 0.017 0.015]]
The mean of the samples was -0.934
```

```
Iteration 68
Acquisition function convergence reached at iteration 969.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.783 0.097 0.029 0.047 0.017 0.027]]
The mean of the samples was -0.472
Iteration 69
Acquisition function convergence reached at iteration 4922.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.801 0.082 0.031 0.047 0.025 0.02 ]]
The mean of the samples was -0.859
Iteration 70
Acquisition function convergence reached at iteration 1569.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.515 0.219 0.031 0.042 0.006 0.015]]
The mean of the samples was -0.836
Iteration 71
Acquisition function convergence reached at iteration 775.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.488 0.2   0.03  0.047 0.011 0.013]]
The mean of the samples was -0.543
Iteration 72
Acquisition function convergence reached at iteration 822.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.744 0.205 0.028 0.052 0.014 0.026]]
The mean of the samples was -0.578
Iteration 73
Acquisition function convergence reached at iteration 1263.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.718 0.153 0.019 0.047 0.017 0.023]]
The mean of the samples was -0.71
Iteration 74
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.794 0.428 0.032 0.037 0.017 0.013]]
The mean of the samples was -1.038
Iteration 75
Acquisition function convergence reached at iteration 10.
The final EI loss was 0.0 with predicted mean of [0.708]
The next parameters to simulate from are [[0.527 0.336 0.03  0.093 0.028 0.043]]
The mean of the samples was 0.781
Iteration 76
Acquisition function convergence reached at iteration 3939.
The final EI loss was -0.399 with predicted mean of [-0.676]
The next parameters to simulate from are [[0.668 0.434 0.029 0.044 0.019 0.028]]
```

```
The mean of the samples was -0.75
Iteration 77
Acquisition function convergence reached at iteration 426.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.725 0.426 0.031 0.044 0.015 0.024]]
The mean of the samples was -0.783
Iteration 78
Acquisition function convergence reached at iteration 1874.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.368 0.643 0.027 0.019 0.034 0.032]]
The mean of the samples was -0.616
Iteration 79
Acquisition function convergence reached at iteration 1223.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.675 0.168 0.024 0.046 0.016 0.01 ]]
The mean of the samples was -0.882
Iteration 80
Acquisition function convergence reached at iteration 201.
The final EI loss was -0.399 with predicted mean of [-0.675]
The next parameters to simulate from are [[0.246 0.838 0.027 0.029 0.015 0.016]]
The mean of the samples was -0.594
Hyperparameter convergence reached at iteration 646.
The minimum predicted mean of the observed indices is -1.343 at the point
[0.596 0.223 0.031 0.037 0.045 0.023]
Iteration 81
Acquisition function convergence reached at iteration 397.
The final EI loss was -0.399 with predicted mean of [-0.672]
The next parameters to simulate from are [[0.6   0.213 0.031 0.034 0.049 0.054]]
The mean of the samples was -0.645
Iteration 82
Acquisition function convergence reached at iteration 409.
The final EI loss was -0.399 with predicted mean of [-0.672]
The next parameters to simulate from are [[0.036 0.372 0.023 0.027 0.013 0.022]]
The mean of the samples was -0.742
Iteration 83
Acquisition function convergence reached at iteration 5092.
The final EI loss was -0.399 with predicted mean of [-0.672]
The next parameters to simulate from are [[0.363 0.07  0.026 0.02  0.033 0.031]]
The mean of the samples was -0.526
Iteration 84
Acquisition function convergence reached at iteration 4782.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.577 0.159 0.027 0.036 0.052 0.015]]
```

```
The mean of the samples was -0.722
Iteration 85
Acquisition function convergence reached at iteration 3905.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.821 0.069 0.029 0.054 0.022 0.015]]
The mean of the samples was -0.823
Iteration 86
Acquisition function convergence reached at iteration 4.
The final EI loss was 0.0 with predicted mean of [-0.856]
The next parameters to simulate from are [[0.672 0.181 0.031 0.056 0.019 0.016]]
The mean of the samples was -0.731
Iteration 87
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [0.019]
The next parameters to simulate from are [[0.503 0.223 0.03  0.069 0.022 0.02 ]]
The mean of the samples was 0.286
Iteration 88
Acquisition function convergence reached at iteration 541.
The final EI loss was -0.399 with predicted mean of [-0.67]
The next parameters to simulate from are [[0.652 0.187 0.013 0.047 0.004 0.02 ]]
The mean of the samples was -0.609
Iteration 89
Acquisition function convergence reached at iteration 6.
The final EI loss was 0.0 with predicted mean of [0.526]
The next parameters to simulate from are [[0.545 0.269 0.031 0.083 0.026 0.045]]
The mean of the samples was 0.432
Iteration 90
Acquisition function convergence reached at iteration 1243.
The final EI loss was -0.399 with predicted mean of [-0.67]
The next parameters to simulate from are [[0.511 0.269 0.032 0.047 0.009 0.014]]
The mean of the samples was -0.482
Iteration 91
Acquisition function convergence reached at iteration 14.
The final EI loss was -0.0 with predicted mean of [-0.005]
The next parameters to simulate from are [[0.395 0.205 0.032 0.052 0.026 0.014]]
The mean of the samples was -0.269
Iteration 92
Acquisition function convergence reached at iteration 927.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.508 0.358 0.032 0.023 0.032 0.029]]
The mean of the samples was -0.593
Iteration 93
Acquisition function convergence reached at iteration 531.
```

```
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.34  0.112 0.026 0.023 0.035 0.045]]
The mean of the samples was -0.837
Iteration 94
Acquisition function convergence reached at iteration 163.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.527 0.303 0.031 0.049 0.02  0.015]]
The mean of the samples was -0.594
Iteration 95
Acquisition function convergence reached at iteration 1179.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.686 0.432 0.032 0.048 0.015 0.029]]
The mean of the samples was -0.71
Iteration 96
Acquisition function convergence reached at iteration 186.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.693 0.225 0.031 0.05  0.014 0.03 ]]
The mean of the samples was -0.717
Iteration 97
Acquisition function convergence reached at iteration 1137.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.75  0.08  0.024 0.047 0.015 0.009]]
The mean of the samples was -0.672
Iteration 98
Acquisition function convergence reached at iteration 697.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.123 0.089 0.027 0.036 0.017 0.025]]
The mean of the samples was -0.907
Iteration 99
Acquisition function convergence reached at iteration 59.
The final EI loss was -0.0 with predicted mean of [-0.83]
The next parameters to simulate from are [[0.696 0.219 0.03  0.048 0.018 0.027]]
The mean of the samples was -0.824
Iteration 100
Acquisition function convergence reached at iteration 68.
The final EI loss was 0.0 with predicted mean of [0.102]
The next parameters to simulate from are [[0.274 0.344 0.032 0.055 0.014 0.012]]
The mean of the samples was 0.219
Hyperparameter convergence reached at iteration 552.
The minimum predicted mean of the observed indices is -1.341 at the point
[0.612 0.223 0.031 0.04  0.022 0.02 ]
Trained parameters:
amplitude_champ:0 is 0.415
```

```
length_scales_champ:0 is [0.25  0.25  0.008 0.019 0.018 0.018]

observation_noise_variance_champ:0 is 0.005

bias_mean:0 is 0.199

Iteration 101
Acquisition function convergence reached at iteration 1204.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.566 0.247 0.032 0.048 0.037 0.015]]
The mean of the samples was -0.443
Iteration 102
Acquisition function convergence reached at iteration 633.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.583 0.208 0.031 0.043 0.023 0.048]]
The mean of the samples was -0.69
Iteration 103
Acquisition function convergence reached at iteration 576.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.484 0.216 0.032 0.044 0.024 0.053]]
The mean of the samples was -0.658
Iteration 104
Acquisition function convergence reached at iteration 5.
The final EI loss was 0.0 with predicted mean of [-0.812]
The next parameters to simulate from are [[0.698 0.21  0.03  0.05  0.017 0.026]]
The mean of the samples was -0.685
Iteration 105
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [-0.013]
The next parameters to simulate from are [[0.612 0.223 0.031 0.068 0.022 0.02 ]]
The mean of the samples was 0.095
Iteration 106
Acquisition function convergence reached at iteration 4613.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.742 0.144 0.026 0.045 0.009 0.009]]
The mean of the samples was -1.008
Iteration 107
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.78]
The next parameters to simulate from are [[0.699 0.212 0.03  0.05  0.017 0.026]]
The mean of the samples was -0.809
Iteration 108
```

```
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.979]
The next parameters to simulate from are [[0.404 0.223 0.031 0.09  0.022 0.029]]
The mean of the samples was 0.906
Iteration 109
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.27]
The next parameters to simulate from are [[0.612 0.223 0.031 0.074 0.022 0.02 ]]
The mean of the samples was 0.294
Iteration 110
WARNING:tensorflow:5 out of the last 4623 calls to <function update_var_EI.<locals>.opt_var a
Acquisition function convergence reached at iteration 148.
The final EI loss was -0.399 with predicted mean of [-0.671]
The next parameters to simulate from are [[0.653 0.442 0.018 0.045 0.019 0.021]]
The mean of the samples was -0.975
Iteration 111
Acquisition function convergence reached at iteration 857.
The final EI loss was -0.399 with predicted mean of [-0.67]
The next parameters to simulate from are [[0.681 0.111 0.018 0.047 0.019 0.027]]
The mean of the samples was -0.728
Iteration 112
Acquisition function convergence reached at iteration 69.
The final EI loss was 0.0 with predicted mean of [-1.046]
The next parameters to simulate from are [[0.611 0.604 0.031 0.04  0.022 0.02 ]]
The mean of the samples was -1.41
Iteration 113
Acquisition function convergence reached at iteration 346.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.273 0.894 0.027 0.027 0.019 0.019]]
The mean of the samples was -0.702
Iteration 114
Acquisition function convergence reached at iteration 3527.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.546 0.373 0.029 0.048 0.02  0.036]]
The mean of the samples was -0.752
Iteration 115
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [-0.071]
The next parameters to simulate from are [[0.612 0.223 0.031 0.013 0.022 0.016]]
The mean of the samples was -0.022
Iteration 116
Acquisition function convergence reached at iteration 593.
The final EI loss was -0.399 with predicted mean of [-0.669]
```

```
The next parameters to simulate from are [[0.583 0.139 0.033 0.044 0.019 0.045]]
The mean of the samples was -0.638
Iteration 117
Acquisition function convergence reached at iteration 16.
The final EI loss was 0.0 with predicted mean of [0.293]
The next parameters to simulate from are [[0.661 0.215 0.031 0.004 0.012 0.005]]
The mean of the samples was 0.26
Iteration 118
Acquisition function convergence reached at iteration 748.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.613 0.192 0.032 0.045 0.071 0.018]]
The mean of the samples was -0.611
Iteration 119
Acquisition function convergence reached at iteration 517.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.323 0.797 0.028 0.031 0.017 0.018]]
The mean of the samples was -0.795
Iteration 120
Acquisition function convergence reached at iteration 123.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.454 0.188 0.032 0.04  0.027 0.055]]
The mean of the samples was -0.63
Hyperparameter convergence reached at iteration 542.
The minimum predicted mean of the observed indices is -1.336 at the point
[0.596 0.223 0.031 0.037 0.045 0.023]
Iteration 121
Acquisition function convergence reached at iteration 136.
The final EI loss was -0.399 with predicted mean of [-0.668]
The next parameters to simulate from are [[0.034 0.113 0.022 0.029 0.012 0.021]]
The mean of the samples was -0.73
Iteration 122
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.852]
The next parameters to simulate from are [[0.178 0.223 0.031 0.082 0.045 0.023]]
The mean of the samples was 1.299
Iteration 123
Acquisition function convergence reached at iteration 598.
The final EI loss was -0.399 with predicted mean of [-0.668]
The next parameters to simulate from are [[0.644 0.208 0.031 0.035 0.052 0.049]]
The mean of the samples was -0.67
Iteration 124
Acquisition function convergence reached at iteration 1256.
The final EI loss was -0.399 with predicted mean of [-0.668]
```

```
The next parameters to simulate from are [[0.303 0.335 0.023 0.02  0.035 0.034]]
The mean of the samples was -0.675
Iteration 125
Acquisition function convergence reached at iteration 134.
The final EI loss was -0.051 with predicted mean of [-1.404]
The next parameters to simulate from are [[0.609 0.129 0.031 0.037 0.041 0.025]]
The mean of the samples was -1.349
Iteration 126
Acquisition function convergence reached at iteration 2286.
The final EI loss was -0.399 with predicted mean of [-0.684]
The next parameters to simulate from are [[0.589 0.195 0.001 0.041 0.026 0.024]]
The mean of the samples was -0.697
Iteration 127
Acquisition function convergence reached at iteration 3513.
The final EI loss was -0.399 with predicted mean of [-0.685]
The next parameters to simulate from are [[0.635 0.333 0.033 0.049 0.023 0.037]]
The mean of the samples was -0.692
Iteration 128
Acquisition function convergence reached at iteration 119.
The final EI loss was -0.003 with predicted mean of [-1.374]
The next parameters to simulate from are [[0.602 0.133 0.031 0.037 0.042 0.026]]
The mean of the samples was -1.407
Iteration 129
Acquisition function convergence reached at iteration 918.
The final EI loss was -0.399 with predicted mean of [-0.693]
The next parameters to simulate from are [[0.807 0.026 0.031 0.049 0.016 0.019]]
The mean of the samples was -0.78
Iteration 130
Acquisition function convergence reached at iteration 1871.
The final EI loss was -0.399 with predicted mean of [-0.693]
The next parameters to simulate from are [[0.579 0.183 0.032 0.032 0.007 0.022]]
The mean of the samples was -0.753
Iteration 131
Acquisition function convergence reached at iteration 919.
The final EI loss was -0.399 with predicted mean of [-0.693]
The next parameters to simulate from are [[0.655 0.151 0.032 0.039 0.044 0.049]]
The mean of the samples was -0.639
Iteration 132
Acquisition function convergence reached at iteration 123.
The final EI loss was -0.0 with predicted mean of [-1.385]
The next parameters to simulate from are [[0.6   0.131 0.031 0.037 0.042 0.026]]
The mean of the samples was -1.37
Iteration 133
```

```
Acquisition function convergence reached at iteration 137.
The final EI loss was -0.003 with predicted mean of [-1.385]
The next parameters to simulate from are [[0.581 0.13  0.031 0.037 0.041 0.027]]
The mean of the samples was -1.161
Iteration 134
Acquisition function convergence reached at iteration 585.
The final EI loss was -0.399 with predicted mean of [-0.668]
The next parameters to simulate from are [[0.137 0.129 0.031 0.033 0.039 0.026]]
The mean of the samples was -0.668
Iteration 135
Acquisition function convergence reached at iteration 162.
The final EI loss was -0.021 with predicted mean of [-1.368]
The next parameters to simulate from are [[0.625 0.259 0.031 0.036 0.05  0.023]]
The mean of the samples was -1.139
Iteration 136
Acquisition function convergence reached at iteration 135.
The final EI loss was -0.015 with predicted mean of [-1.357]
The next parameters to simulate from are [[0.594 0.172 0.031 0.037 0.043 0.024]]
The mean of the samples was -1.089
Iteration 137
Acquisition function convergence reached at iteration 5.
The final EI loss was 0.0 with predicted mean of [-0.814]
The next parameters to simulate from are [[0.702 0.241 0.03  0.049 0.017 0.025]]
The mean of the samples was -0.828
Iteration 138
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.798]
The next parameters to simulate from are [[0.699 0.227 0.03  0.049 0.017 0.026]]
The mean of the samples was -0.796
Iteration 139
Acquisition function convergence reached at iteration 1167.
The final EI loss was -0.399 with predicted mean of [-0.669]
The next parameters to simulate from are [[0.347 0.159 0.031 0.036 0.025 0.019]]
The mean of the samples was -0.742
Iteration 140
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.394]
The next parameters to simulate from are [[0.612 0.223 0.031 0.078 0.022 0.02 ]]
The mean of the samples was 0.428
Hyperparameter convergence reached at iteration 563.
The minimum predicted mean of the observed indices is -1.333 at the point
[0.612 0.223 0.031 0.04  0.022 0.02 ]
Iteration 141
```

```
Acquisition function convergence reached at iteration 158.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.601 0.131 0.006 0.034 0.059 0.021]]
The mean of the samples was -0.975
Iteration 142
Acquisition function convergence reached at iteration 136.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.693 0.112 0.03  0.032 0.01  0.014]]
The mean of the samples was -0.836
Iteration 143
Acquisition function convergence reached at iteration 3224.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.507 0.192 0.032 0.035 0.05  0.06 ]]
The mean of the samples was -0.716
Iteration 144
Acquisition function convergence reached at iteration 3320.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.186 0.137 0.033 0.032 0.036 0.021]]
The mean of the samples was -0.652
Iteration 145
Acquisition function convergence reached at iteration 1482.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.109 0.16  0.031 0.031 0.032 0.021]]
The mean of the samples was -0.613
Iteration 146
Acquisition function convergence reached at iteration 275.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.647 0.708 0.018 0.046 0.016 0.024]]
The mean of the samples was -0.674
Iteration 147
Acquisition function convergence reached at iteration 5.
The final EI loss was 0.0 with predicted mean of [-0.803]
The next parameters to simulate from are [[0.696 0.238 0.03  0.049 0.017 0.026]]
The mean of the samples was -0.797
Iteration 148
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [-0.25]
The next parameters to simulate from are [[0.896 0.223 0.031 0.04  0.022 0.02 ]]
The mean of the samples was -0.245
Iteration 149
Acquisition function convergence reached at iteration 1025.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.528 0.181 0.    0.043 0.024 0.018]]
```

```
The mean of the samples was -0.889
Iteration 150
Acquisition function convergence reached at iteration 9392.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.819 0.018 0.033 0.048 0.018 0.014]]
The mean of the samples was -1.007
Trained parameters:
amplitude_champ:0 is 0.389

length_scales_champ:0 is [0.25  0.25  0.008 0.02  0.018 0.018]

observation_noise_variance_champ:0 is 0.006

bias_mean:0 is 0.201

Iteration 151
Acquisition function convergence reached at iteration 1293.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.593 0.066 0.028 0.041 0.022 0.011]]
The mean of the samples was -0.509
Iteration 152
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.797]
The next parameters to simulate from are [[0.697 0.262 0.03  0.049 0.017 0.026]]
The mean of the samples was -0.78
Iteration 153
Acquisition function convergence reached at iteration 709.
The final EI loss was -0.399 with predicted mean of [-0.666]
The next parameters to simulate from are [[0.726 0.906 0.031 0.038 0.019 0.018]]
The mean of the samples was -0.797
Iteration 154
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.243]
The next parameters to simulate from are [[0.612 0.223 0.031 0.006 0.022 0.02 ]]
The mean of the samples was 0.142
Iteration 155
Acquisition function convergence reached at iteration 7.
The final EI loss was -0.0 with predicted mean of [-0.205]
The next parameters to simulate from are [[0.331 0.303 0.031 0.047 0.018 0.016]]
The mean of the samples was -0.324
Iteration 156
Acquisition function convergence reached at iteration 393.
The final EI loss was -0.399 with predicted mean of [-0.666]
```

```
The next parameters to simulate from are [[0.405 0.92  0.028 0.023 0.036 0.031]]
The mean of the samples was -0.764
Iteration 157
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.376]
The next parameters to simulate from are [[0.682 0.223 0.031 0.002 0.022 0.02 ]]
The mean of the samples was 0.294
Iteration 158
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.794]
The next parameters to simulate from are [[0.698 0.265 0.03  0.049 0.017 0.026]]
The mean of the samples was -0.874
Iteration 159
Acquisition function convergence reached at iteration 158.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.679 0.122 0.015 0.05  0.019 0.025]]
The mean of the samples was -0.564
Iteration 160
Acquisition function convergence reached at iteration 133.
The final EI loss was -0.004 with predicted mean of [-1.338]
The next parameters to simulate from are [[0.59  0.589 0.03  0.038 0.034 0.021]]
The mean of the samples was -1.29
Hyperparameter convergence reached at iteration 541.
The minimum predicted mean of the observed indices is -1.331 at the point
[0.612 0.223 0.031 0.04  0.022 0.02 ]
Iteration 161
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.793]
The next parameters to simulate from are [[0.698 0.266 0.03  0.049 0.017 0.026]]
The mean of the samples was -0.79
Iteration 162
Acquisition function convergence reached at iteration 3968.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.722 0.009 0.021 0.047 0.017 0.021]]
The mean of the samples was -0.66
Iteration 163
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.795]
The next parameters to simulate from are [[0.698 0.268 0.031 0.05  0.017 0.026]]
The mean of the samples was -0.854
Iteration 164
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.858]
```

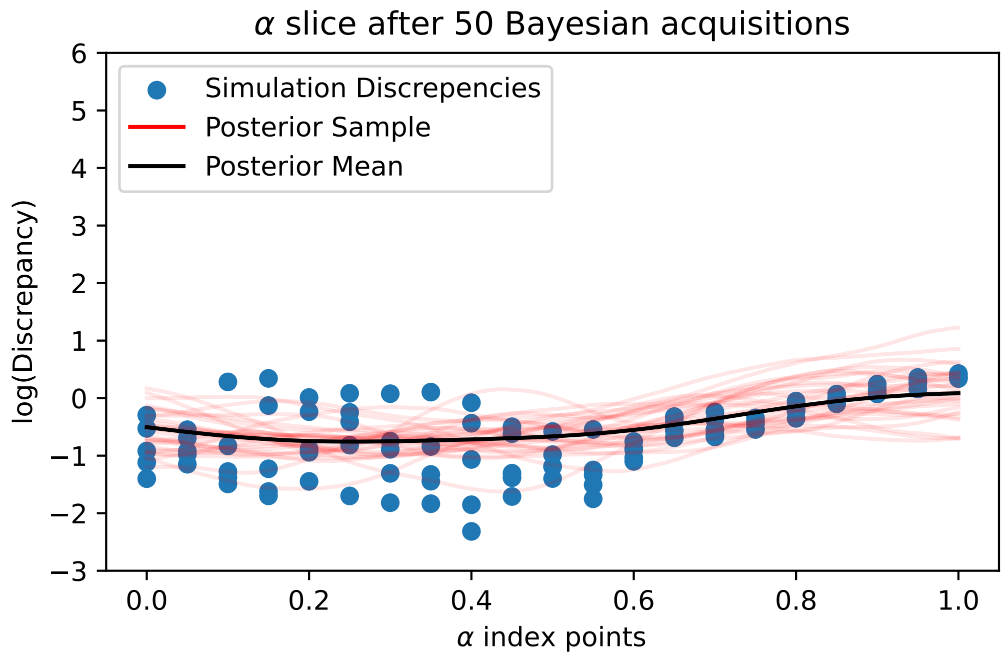
```
The next parameters to simulate from are [[0.612 0.909 0.031 0.1   0.022 0.02 ]]
The mean of the samples was 0.948
Iteration 165
Acquisition function convergence reached at iteration 146.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.563 0.172 0.032 0.043 0.064 0.018]]
The mean of the samples was -0.596
Iteration 166
Acquisition function convergence reached at iteration 7.
The final EI loss was -0.0 with predicted mean of [-0.78]
The next parameters to simulate from are [[0.711 0.259 0.03  0.052 0.015 0.023]]
The mean of the samples was -0.718
Iteration 167
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [-0.298]
The next parameters to simulate from are [[0.874 0.223 0.029 0.04  0.022 0.02 ]]
The mean of the samples was -0.358
Iteration 168
Acquisition function convergence reached at iteration 4808.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.769 0.107 0.033 0.051 0.034 0.013]]
The mean of the samples was -1.051
Iteration 169
Acquisition function convergence reached at iteration 1760.
The final EI loss was -0.399 with predicted mean of [-0.664]
The next parameters to simulate from are [[0.762 0.905 0.032 0.041 0.013 0.015]]
The mean of the samples was -1.016
Iteration 170
Acquisition function convergence reached at iteration 522.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.703 0.109 0.025 0.043 0.04  0.007]]
The mean of the samples was -0.491
Iteration 171
Acquisition function convergence reached at iteration 2.
The final EI loss was -0.0 with predicted mean of [-0.157]
The next parameters to simulate from are [[0.554 0.223 0.031 0.061 0.022 0.02 ]]
The mean of the samples was -0.11
Iteration 172
Acquisition function convergence reached at iteration 1079.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.785 0.041 0.033 0.058 0.016 0.011]]
The mean of the samples was -0.936
Iteration 173
```

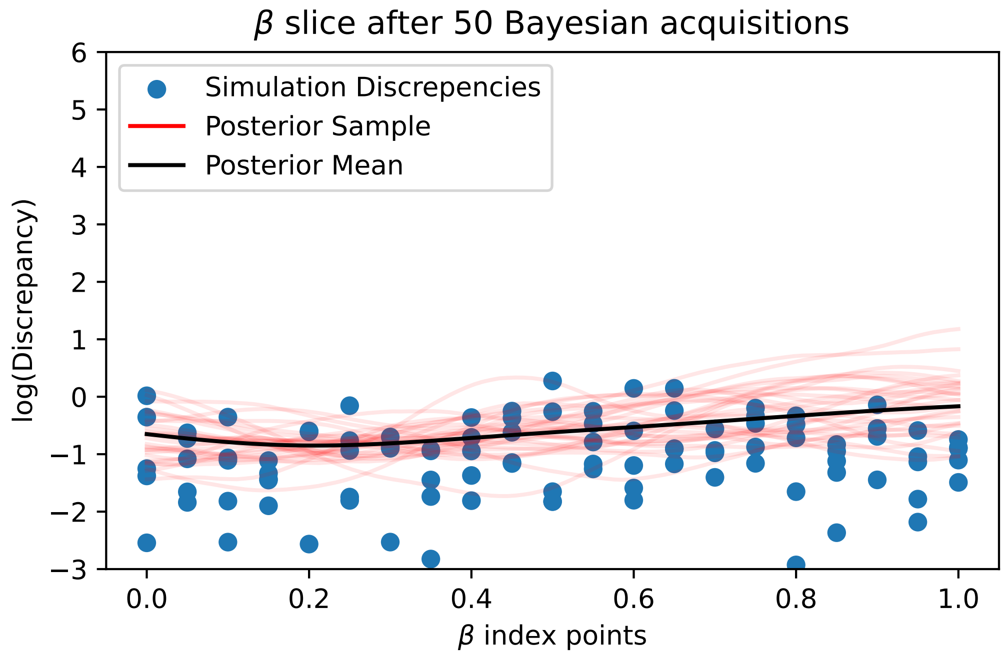
```
Acquisition function convergence reached at iteration 747.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.17  0.253 0.032 0.03  0.027 0.02 ]]
The mean of the samples was -0.619
Iteration 174
Acquisition function convergence reached at iteration 1124.
The final EI loss was -0.397 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.903 0.584 0.016 0.033 0.066 0.003]]
The mean of the samples was -1.312
Iteration 175
Acquisition function convergence reached at iteration 5.
The final EI loss was 0.0 with predicted mean of [-0.797]
The next parameters to simulate from are [[0.701 0.266 0.031 0.05  0.017 0.025]]
The mean of the samples was -0.819
Iteration 176
Acquisition function convergence reached at iteration 1072.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.612 0.236 0.029 0.034 0.029 0.043]]
The mean of the samples was -0.658
Iteration 177
Acquisition function convergence reached at iteration 1512.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.751 0.422 0.031 0.057 0.02  0.014]]
The mean of the samples was -0.873
Iteration 178
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.486]
The next parameters to simulate from are [[0.612 0.223 0.033 0.084 0.022 0.02 ]]
The mean of the samples was 0.508
Iteration 179
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.042]
The next parameters to simulate from are [[0.612 0.223 0.031 0.068 0.022 0.021]]
The mean of the samples was -0.09
Iteration 180
Acquisition function convergence reached at iteration 129.
The final EI loss was -0.003 with predicted mean of [-1.336]
The next parameters to simulate from are [[0.655 0.469 0.03  0.042 0.021 0.015]]
The mean of the samples was -1.344
Hyperparameter convergence reached at iteration 548.
The minimum predicted mean of the observed indices is -1.333 at the point
[0.655 0.469 0.03  0.042 0.021 0.015]
Iteration 181
```

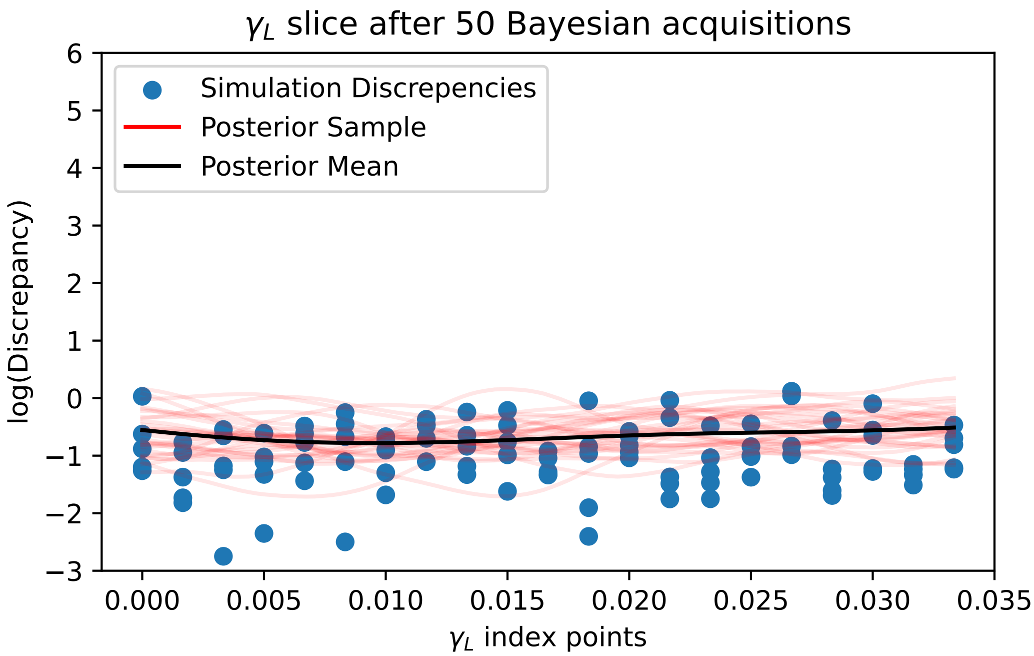
```
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.736]
The next parameters to simulate from are [[0.655 0.469 0.03  0.099 0.021 0.015]]
The mean of the samples was 0.933
Iteration 182
Acquisition function convergence reached at iteration 3502.
The final EI loss was -0.399 with predicted mean of [-0.667]
The next parameters to simulate from are [[0.512 0.626 0.032 0.042 0.024 0.047]]
The mean of the samples was -0.655
Iteration 183
Acquisition function convergence reached at iteration 666.
The final EI loss was -0.399 with predicted mean of [-0.667]
The next parameters to simulate from are [[0.664 0.868 0.031 0.052 0.019 0.021]]
The mean of the samples was -0.493
Iteration 184
Acquisition function convergence reached at iteration 110.
The final EI loss was -0.002 with predicted mean of [-1.337]
The next parameters to simulate from are [[0.674 0.467 0.03  0.043 0.021 0.014]]
The mean of the samples was -1.467
Iteration 185
Acquisition function convergence reached at iteration 1793.
The final EI loss was -0.399 with predicted mean of [-0.691]
The next parameters to simulate from are [[0.34  0.298 0.024 0.019 0.042 0.041]]
The mean of the samples was -0.753
Iteration 186
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.232]
The next parameters to simulate from are [[0.674 0.467 0.03  0.004 0.022 0.014]]
The mean of the samples was 0.238
Iteration 187
The final EI loss was -0.399 with predicted mean of [-0.692]
The next parameters to simulate from are [[0.582 0.393 0.001 0.043 0.021 0.016]]
The mean of the samples was -0.743
Iteration 188
Acquisition function convergence reached at iteration 119.
The final EI loss was -0.005 with predicted mean of [-1.391]
The next parameters to simulate from are [[0.682 0.474 0.03  0.043 0.02  0.013]]
The mean of the samples was -1.149
Iteration 189
Acquisition function convergence reached at iteration 600.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.529 0.115 0.03  0.036 0.045 0.064]]
The mean of the samples was -0.667
```

```
Iteration 190
Acquisition function convergence reached at iteration 926.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.101 0.263 0.031 0.035 0.017 0.024]]
The mean of the samples was -0.725
Iteration 191
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.796]
The next parameters to simulate from are [[0.701 0.265 0.031 0.05  0.017 0.026]]
The mean of the samples was -0.774
Iteration 192
Acquisition function convergence reached at iteration 680.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.036 0.126 0.029 0.034 0.04  0.03 ]]
The mean of the samples was -0.773
Iteration 193
Acquisition function convergence reached at iteration 182.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.506 0.116 0.03  0.045 0.03  0.017]]
The mean of the samples was -0.699
Iteration 194
Acquisition function convergence reached at iteration 2.
The final EI loss was 0.0 with predicted mean of [0.7]
The next parameters to simulate from are [[0.612 0.223 0.015 0.091 0.022 0.02 ]]
The mean of the samples was 0.797
Iteration 195
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.788]
The next parameters to simulate from are [[0.701 0.265 0.031 0.05  0.017 0.026]]
The mean of the samples was -0.803
Iteration 196
Acquisition function convergence reached at iteration 1189.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.694 0.471 0.03  0.048 0.066 0.011]]
The mean of the samples was -0.512
Iteration 197
Acquisition function convergence reached at iteration 474.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.581 0.154 0.031 0.051 0.02  0.04 ]]
The mean of the samples was -0.671
Iteration 198
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.772]
```

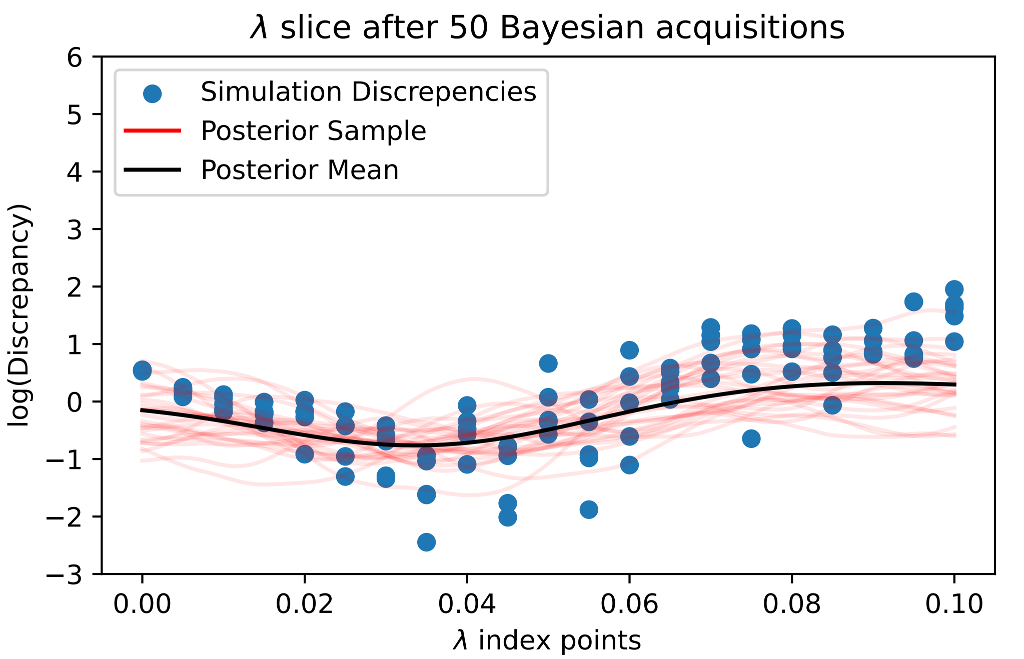
```
The next parameters to simulate from are [[0.699 0.266 0.03  0.05  0.017 0.026]]
The mean of the samples was -0.752
Iteration 199
Acquisition function convergence reached at iteration 160.
The final EI loss was -0.399 with predicted mean of [-0.665]
The next parameters to simulate from are [[0.311 0.062 0.031 0.041 0.022 0.021]]
The mean of the samples was -0.761
Iteration 200
Acquisition function convergence reached at iteration 5.
The final EI loss was -0.0 with predicted mean of [-0.763]
The next parameters to simulate from are [[0.7   0.268 0.03  0.05  0.017 0.026]]
The mean of the samples was -0.693
Hyperparameter convergence reached at iteration 541.
The minimum predicted mean of the observed indices is -1.329 at the point
[0.612 0.223 0.031 0.04  0.022 0.02 ]
Trained parameters:
amplitude_champ:0 is 0.376

length_scales_champ:0 is [0.25  0.25  0.008 0.02  0.018 0.018]

observation_noise_variance_champ:0 is 0.005

bias_mean:0 is 0.194
```


α slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
α index points

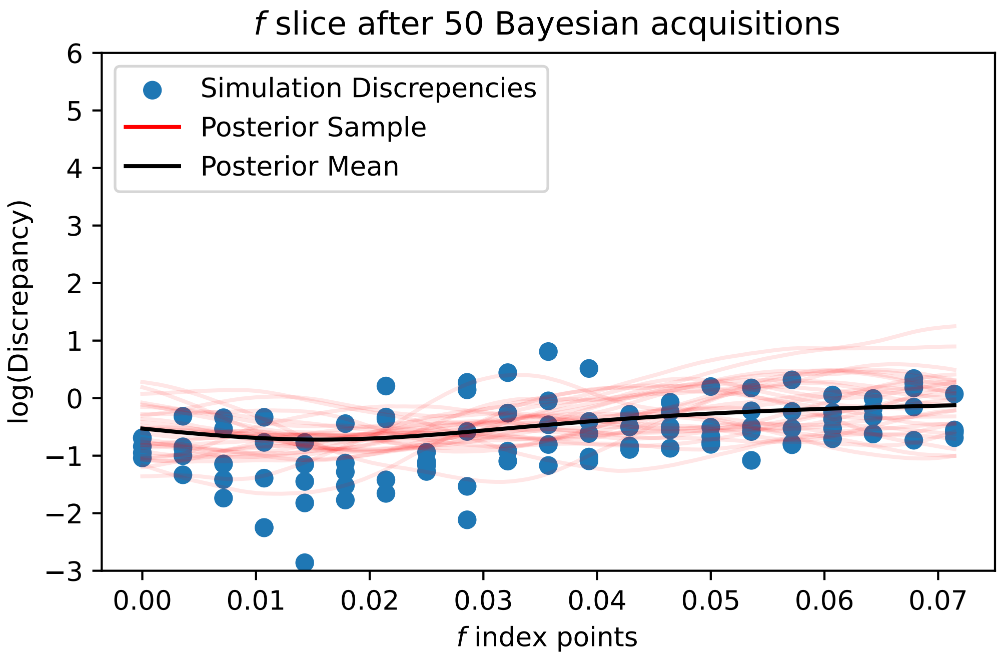

β slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
β index points



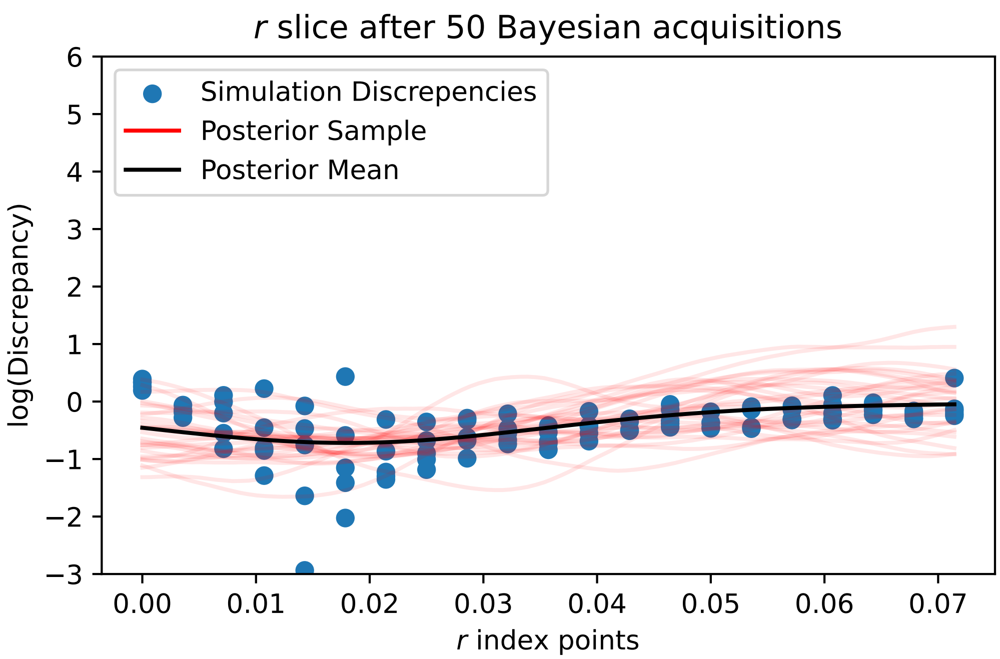
γ_L slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
6
5
4
3
2
1
0
−1
−2
−3
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
γ_L index points



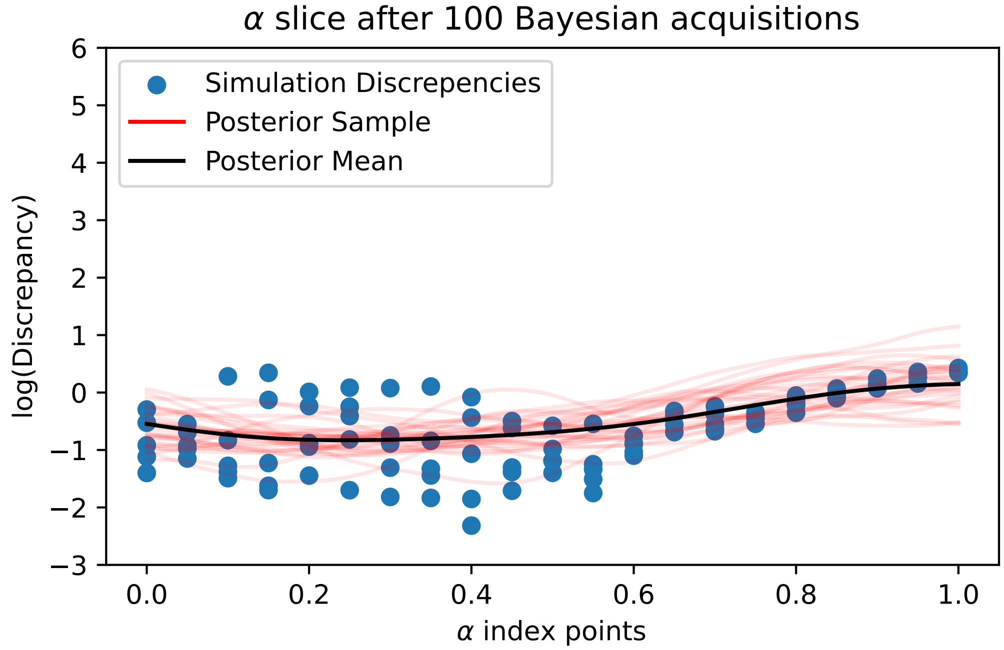
λ slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
6
5
4
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
λ index points

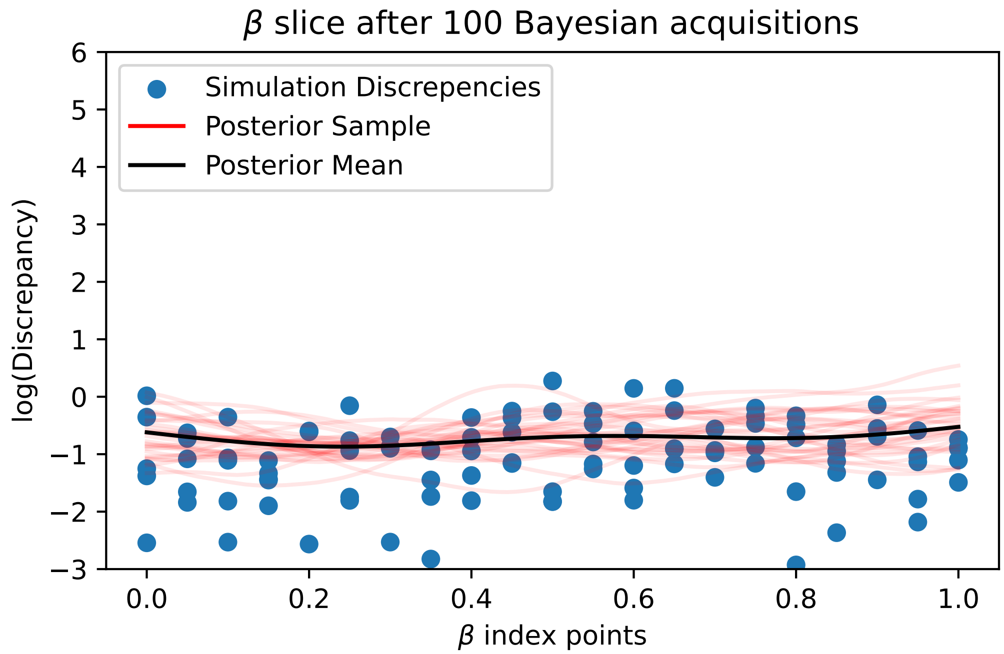


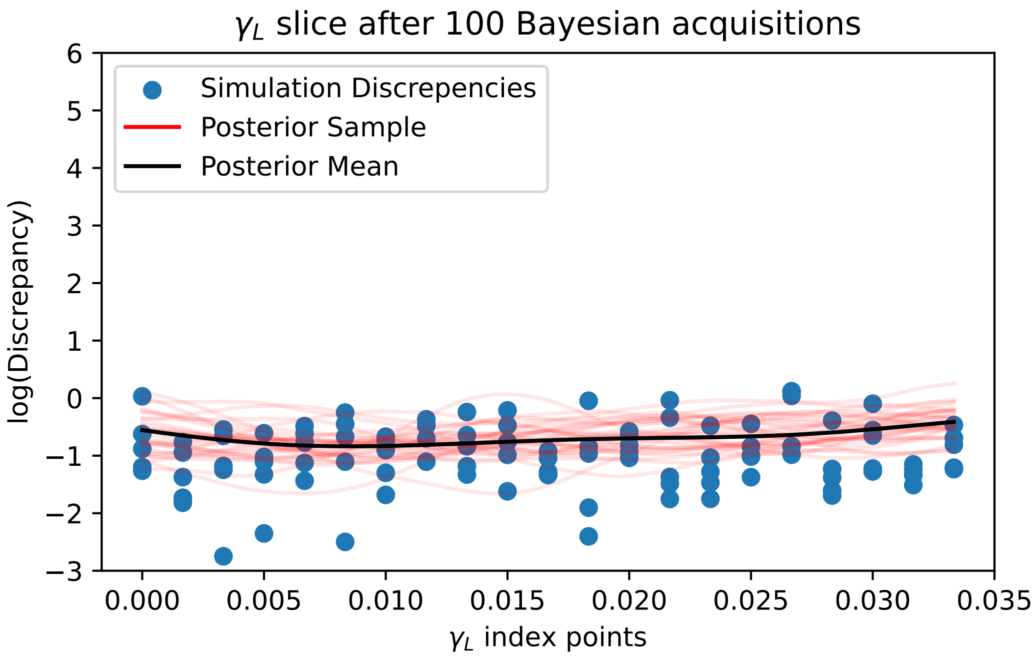
f slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
f index points

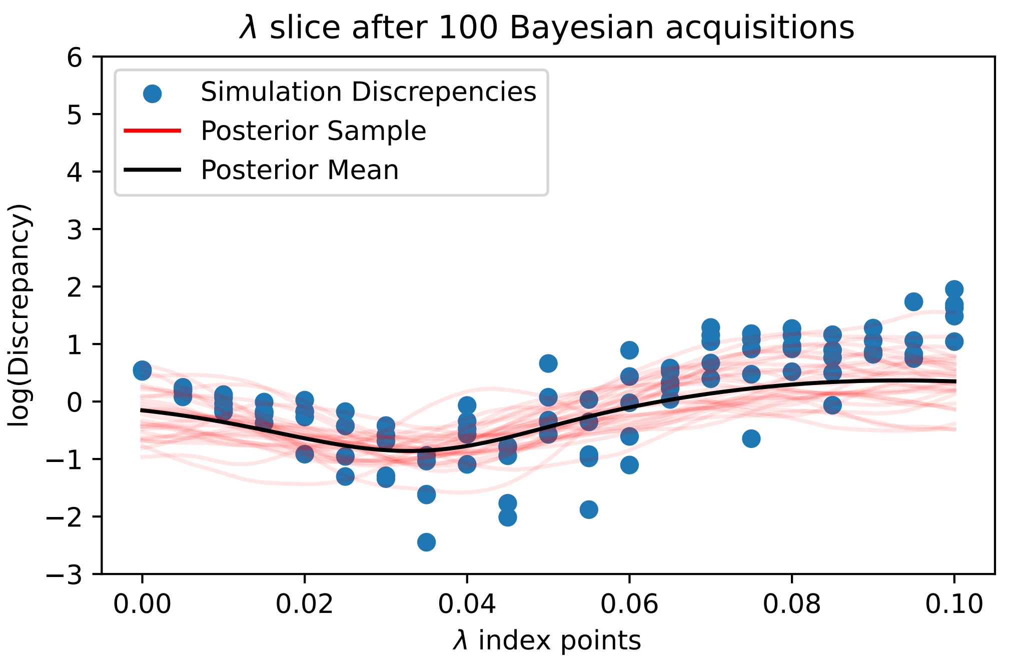


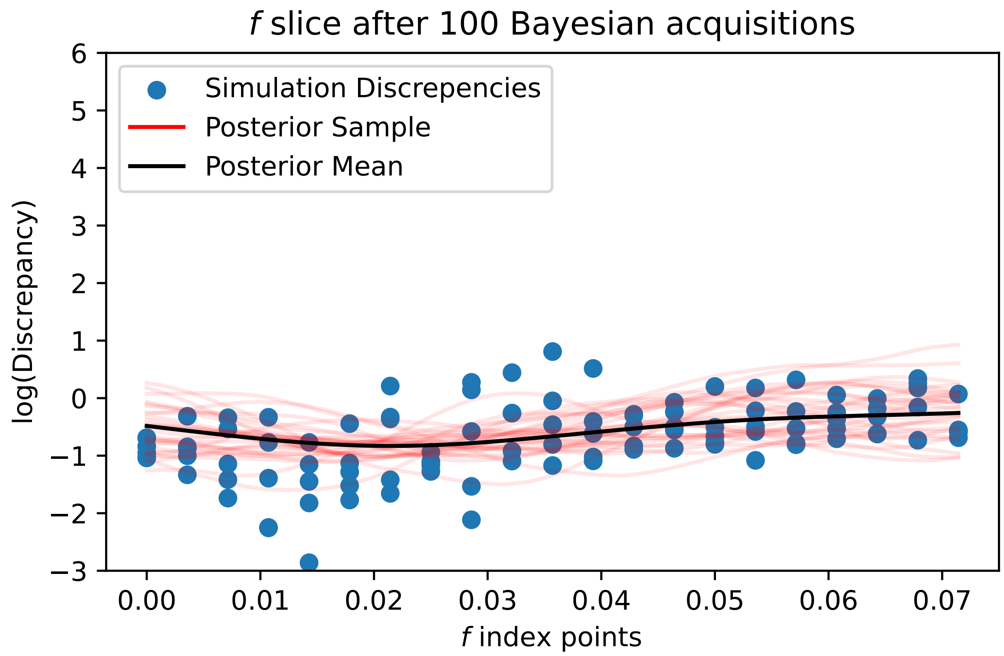
r slice after 50 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
r index points

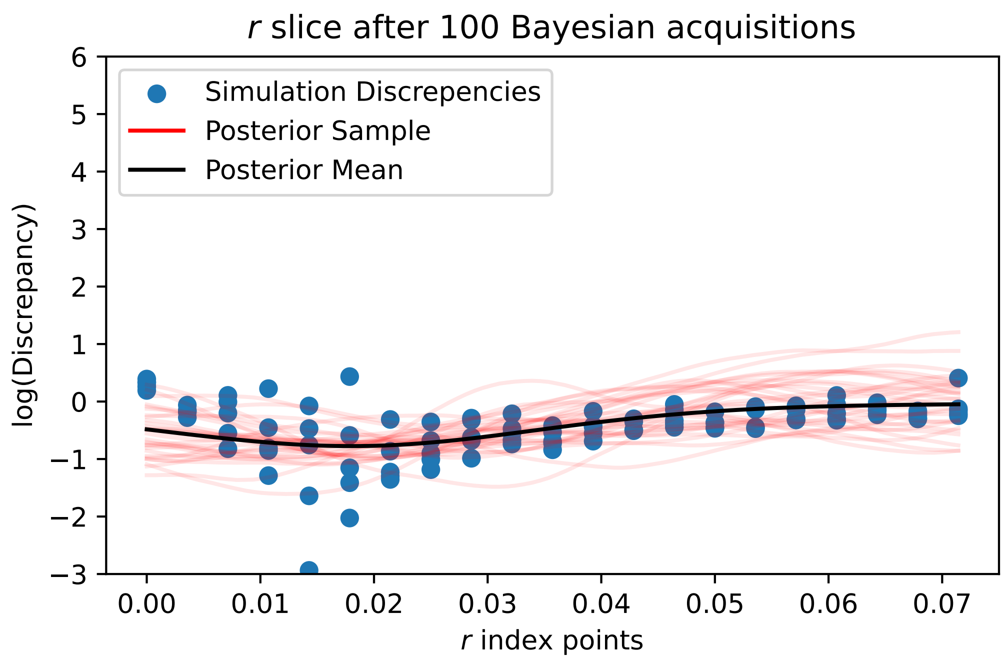

α slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
α index points

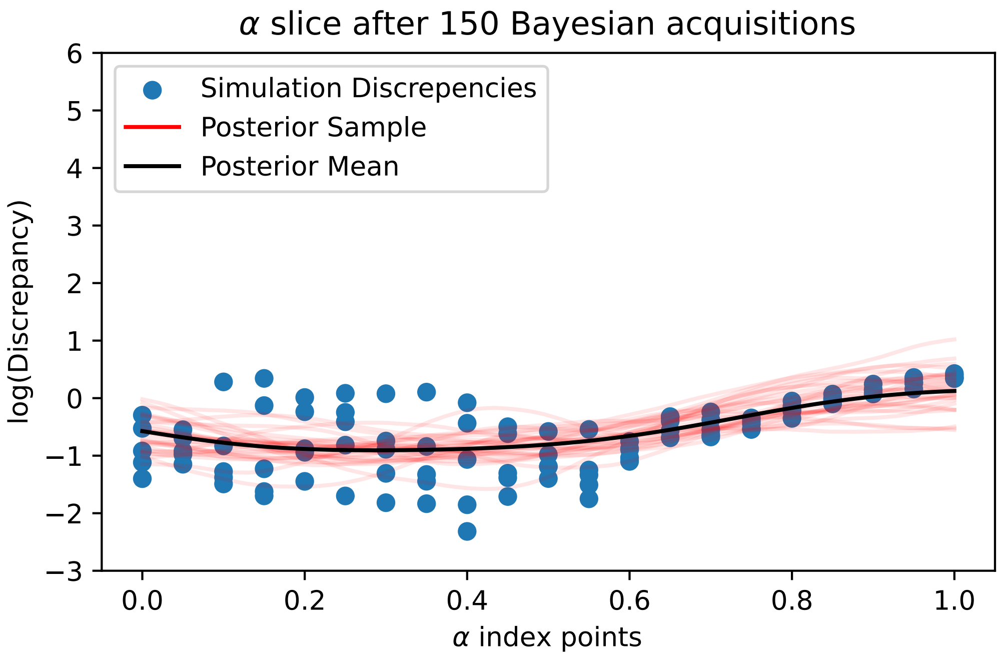

β slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
β index points

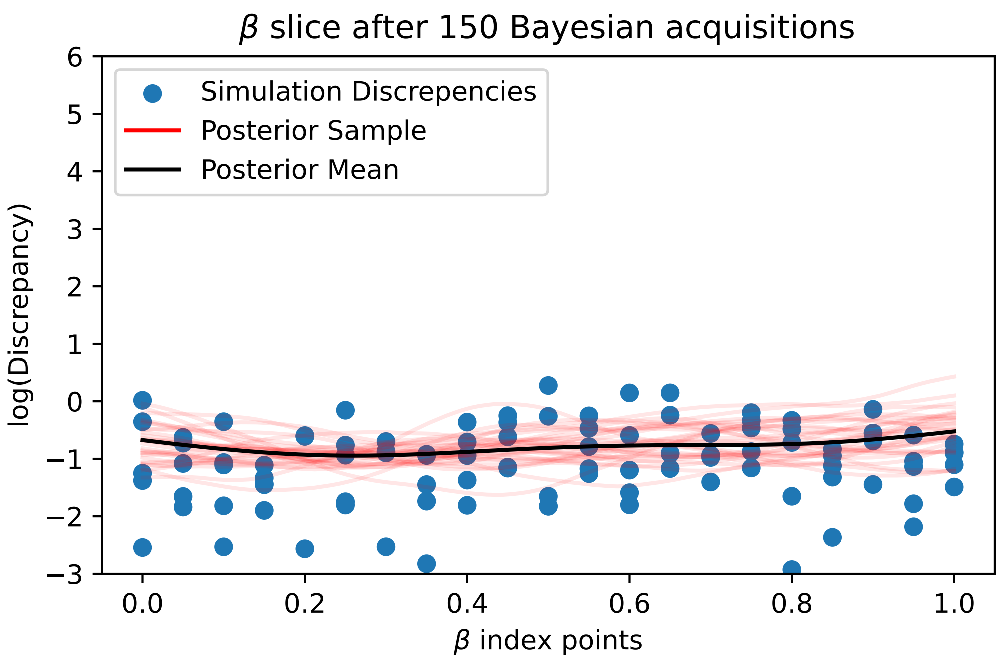

γ_L slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
γ_L index points
6
5
4
3
2
1
0
−1
−2
−3
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035

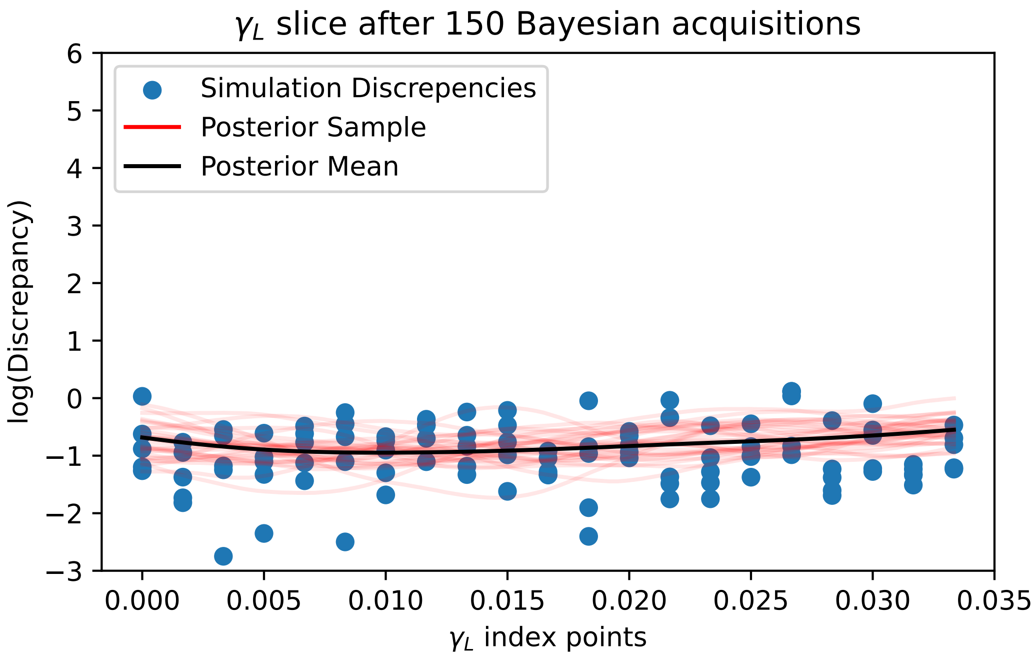

λ slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
λ index points
6
5
4
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10


f slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
f index points

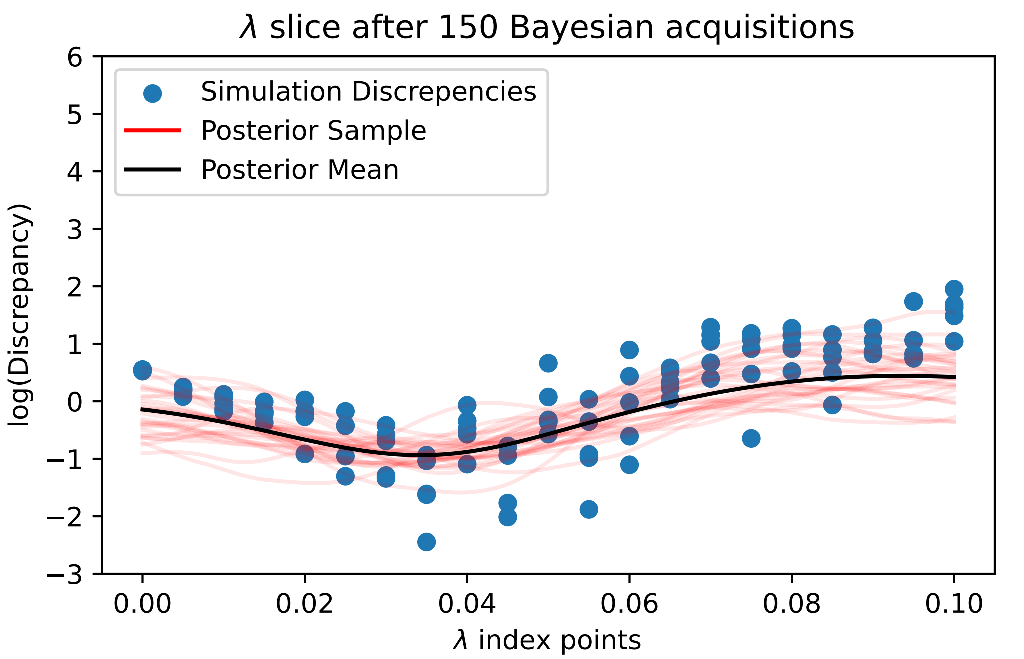

r slice after 100 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
r index points


α slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
α index points

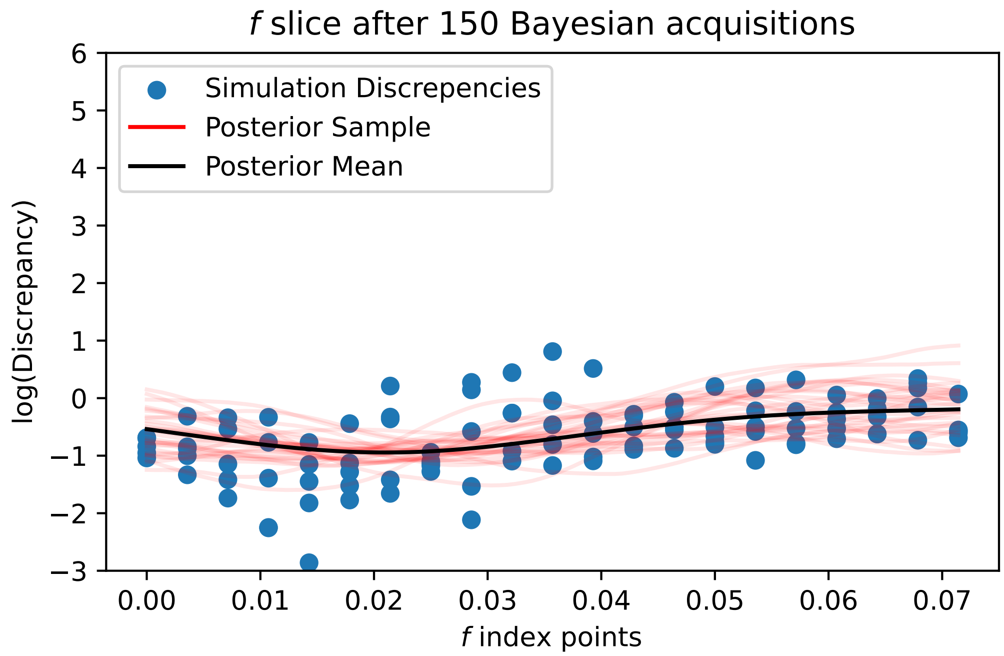

β slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
β index points

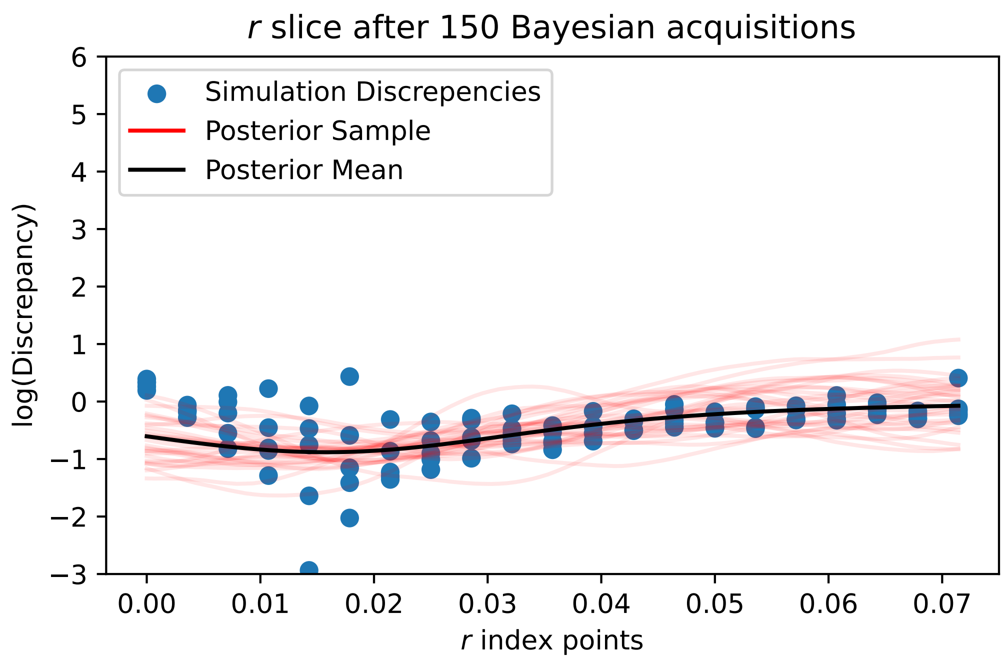


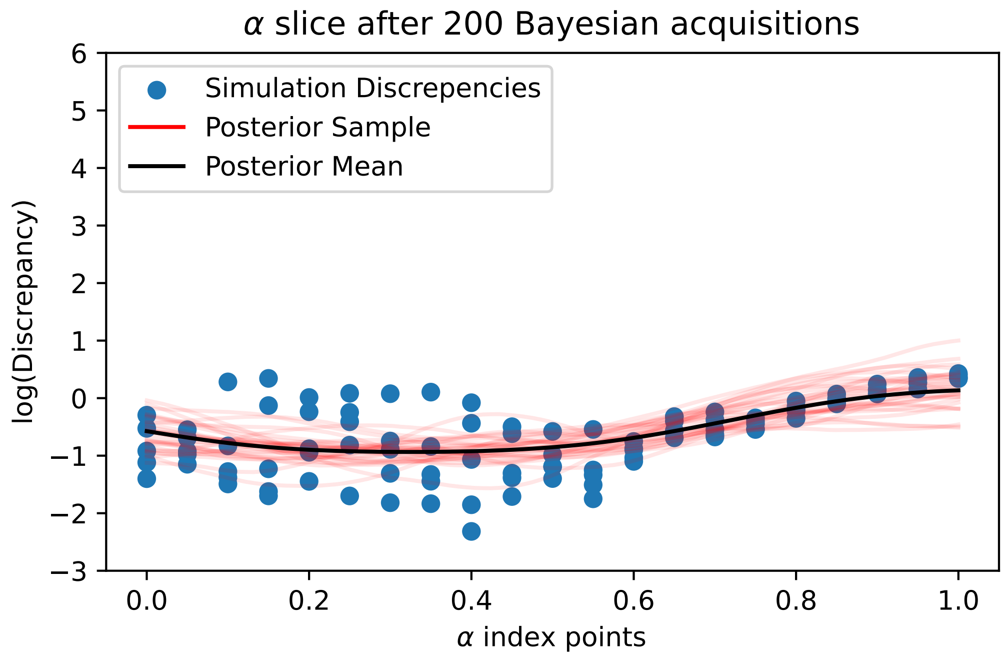
γ_L slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
6
5
4
3
2
1
0
−1
−2
−3
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
γ_L index points

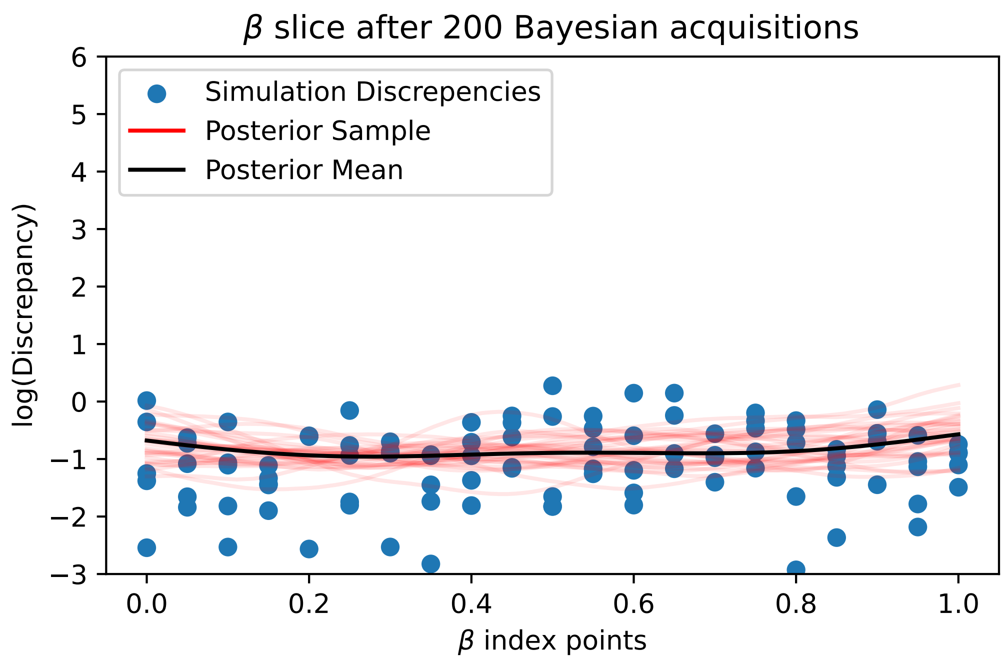


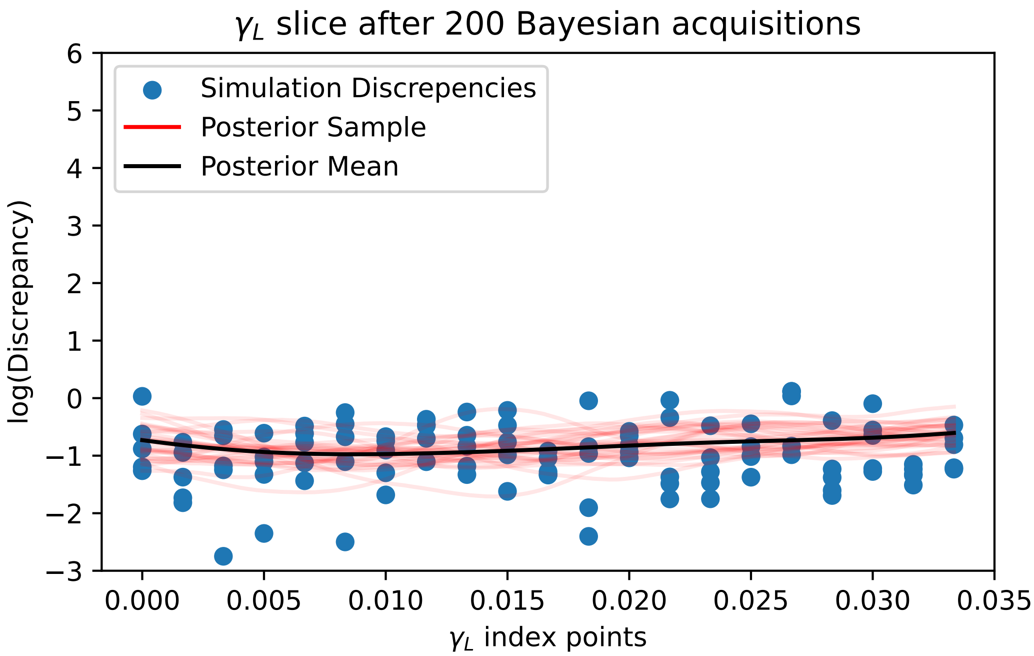
λ slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
6
5
4
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
λ index points

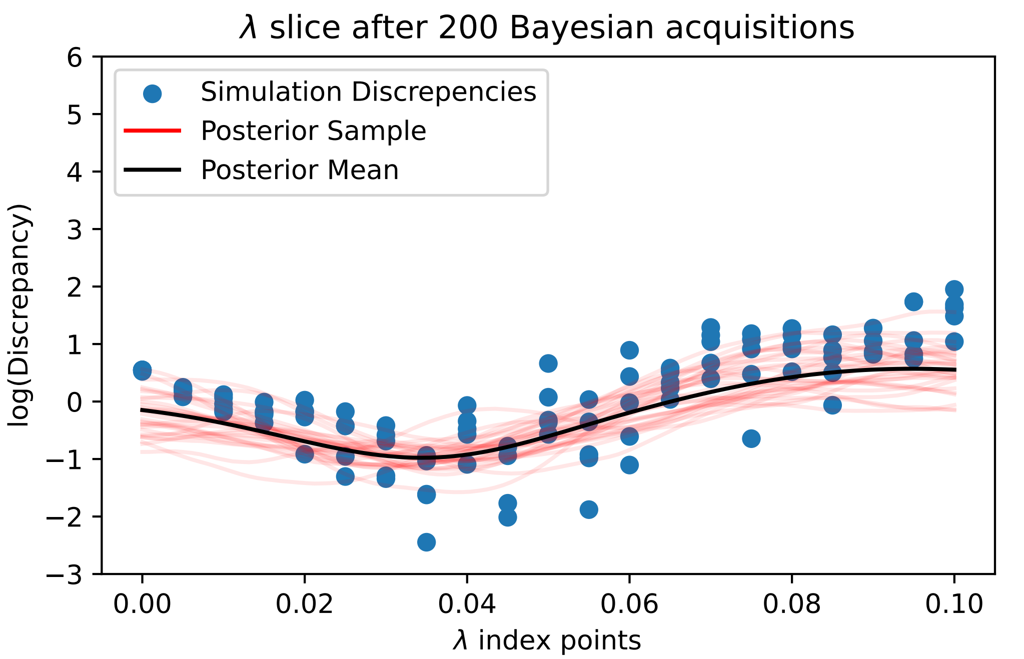

f slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
f index points

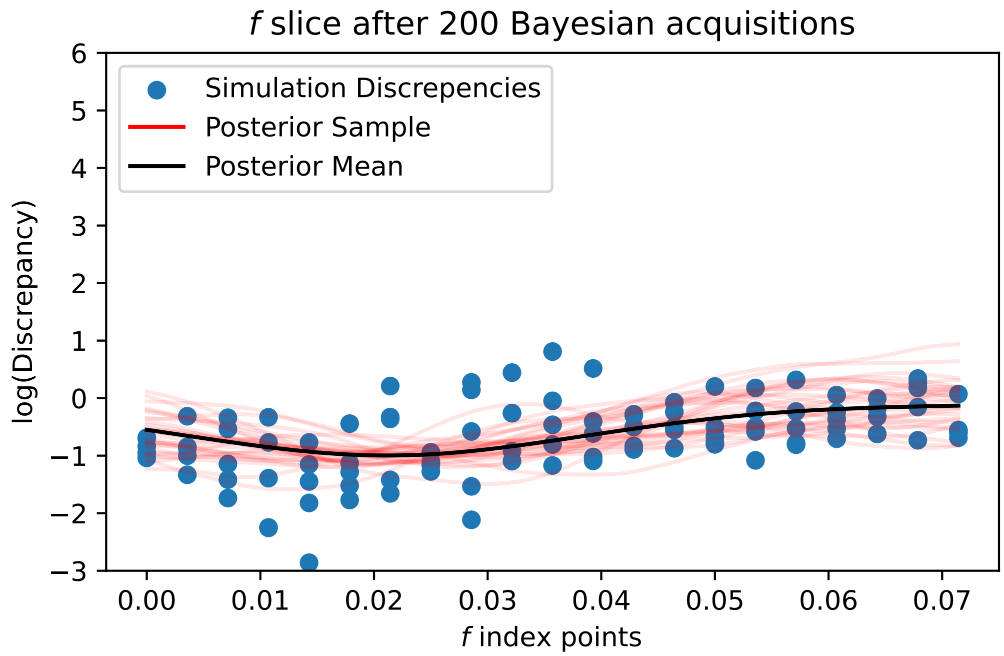

r slice after 150 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
r index points

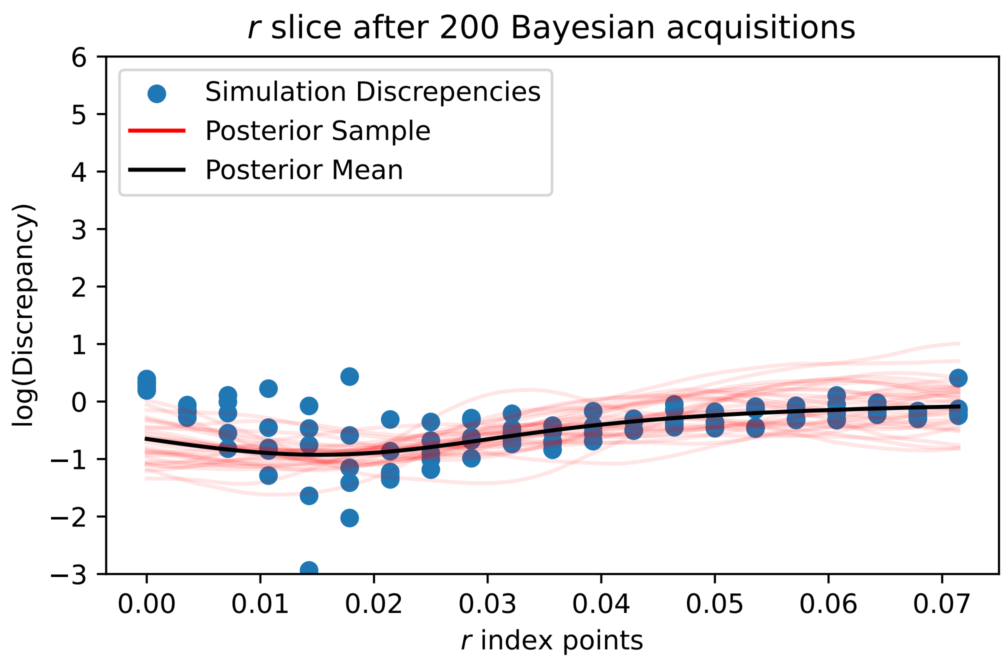

α slice after 200 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
α index points


β slice after 200 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
β index points


γ_L slice after 200 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
γ_L index points
6
5
4
3
2
1
0
−1
−2
−3
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035

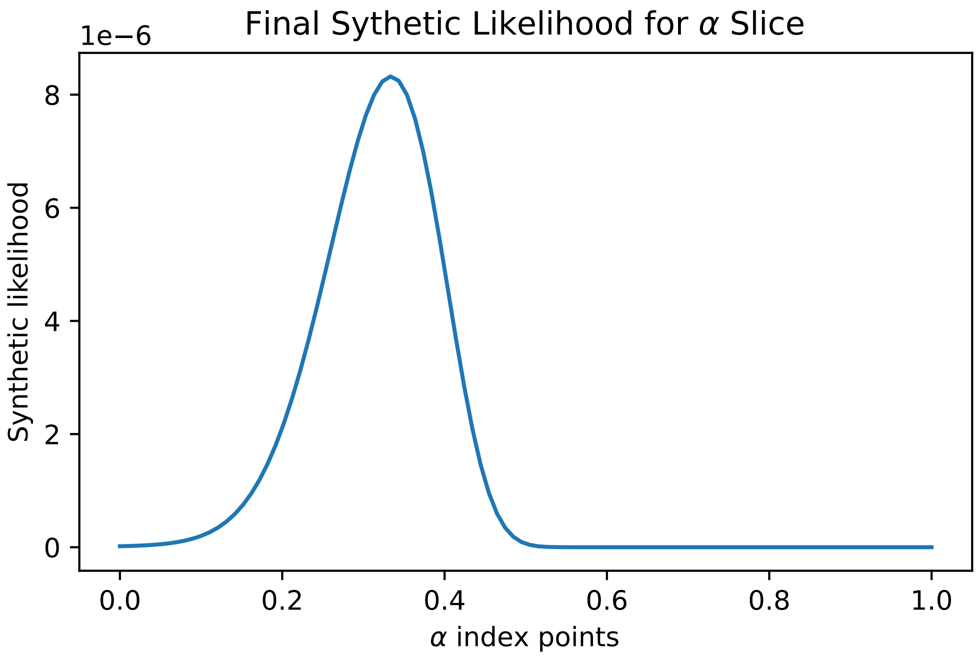

λ slice after 200 Bayesian acquisitions
Simulation Discrepencies
Posterior Sample
Posterior Mean
log(Discrepancy)
λ index points
6
5
4
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10

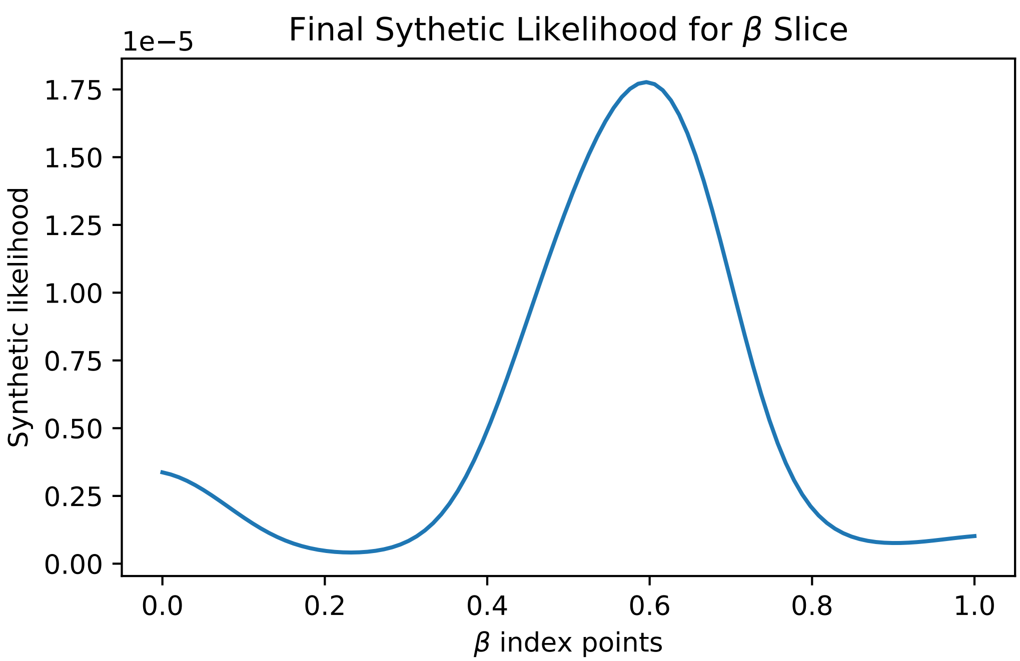


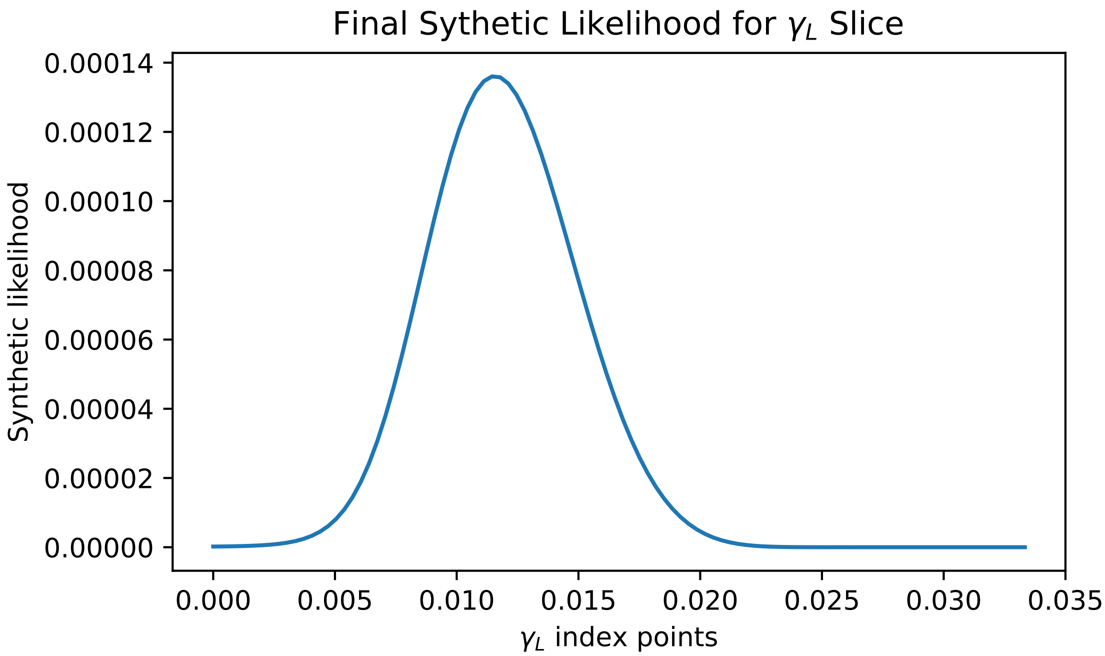


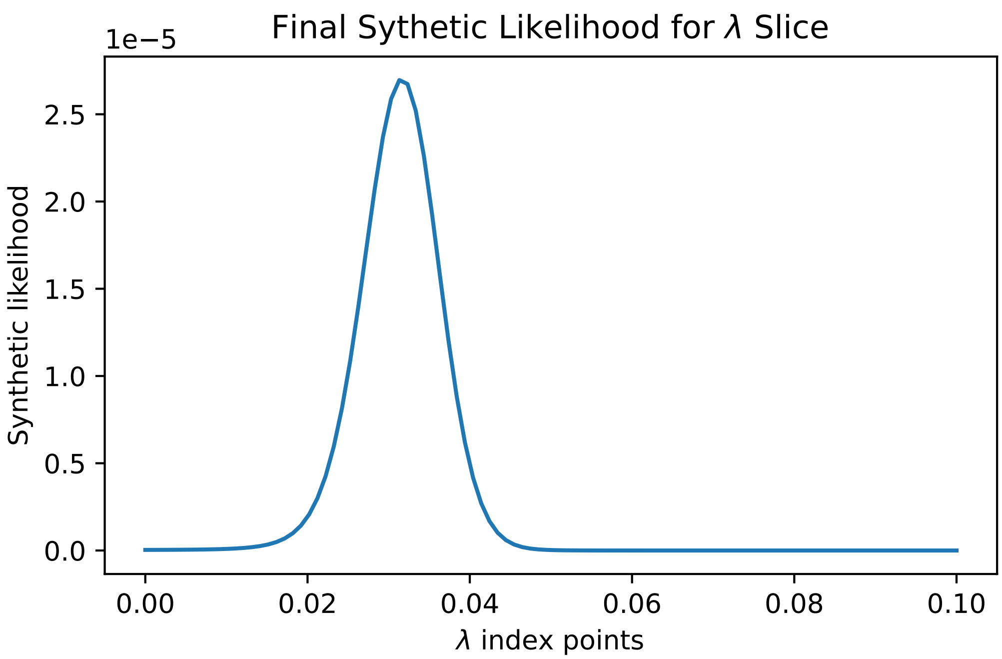

```
epsilon = -2.
for var in vars:
    champ_GP_reg = tfd.GaussianProcessRegressionModel(
```

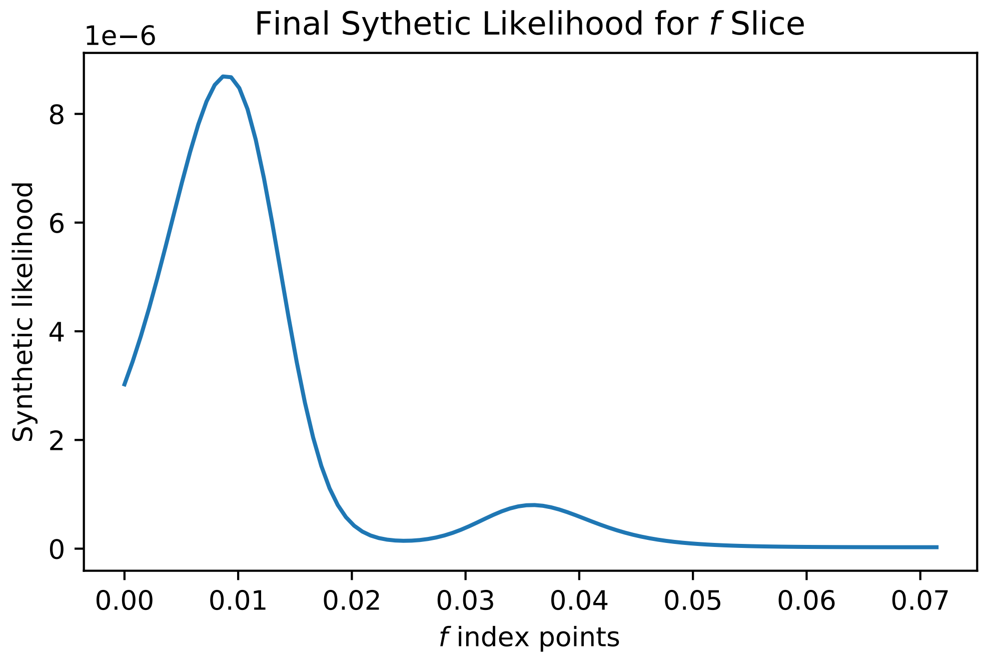
```
        kernel=kernel_champ,
        index_points=slice_indices_dfs_dict[var + "_gp_indices_df"].values,
        observation_index_points=index_vals,
        observations=obs_vals,
        observation_noise_variance=observation_noise_variance_champ,
        predictive_noise_variance=0.0,
        mean_fn=const_mean_fn(),
    )

    indices_for_lik = slice_indices_dfs_dict[var + "_gp_indices_df"].values

    mean = champ_GP_reg.mean_fn(indices_for_lik)
    variance = champ_GP_reg.variance(index_points=indices_for_lik)
    post_std = np.sqrt(variance)
    cdf_vals = tfd.Normal(mean, post_std).log_cdf(epsilon)

    plt.figure(figsize=(6, 3.5))
    plt.plot(
        slice_indices_dfs_dict[var + "_gp_indices_df"][var].values,
        np.exp(cdf_vals),
    )
    if var in ["f", "r"]:
        plt.xlabel("$" + var + "$ index points")
        plt.title("Final Sythetic Likelihood for $" + var + "$ Slice")
    else:
        plt.xlabel("$\\" + var + "$ index points")
        plt.title("Final Sythetic Likelihood for $\\" + var + "$ Slice")
    plt.ylabel("Synthetic likelihood")
    plt.savefig(
        "champagne_GP_images/"
        + var
        + "_slice_"
        + str(t)
        + "_synth_likelihood.pdf"
    )
    plt.show()
```

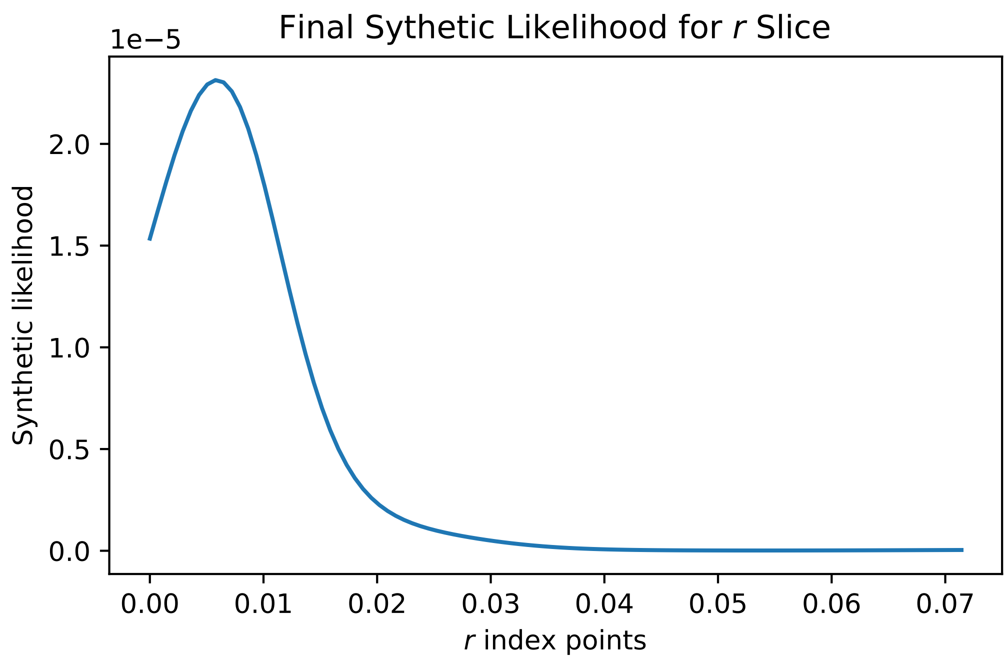

Final Sythetic Likelihood for α Slice
1e−6
8
6
4
2
0
Synthetic likelihood
0.0
0.2
0.4
0.6
0.8
1.0
α index points


Final Sythetic Likelihood for β Slice
1e−5
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Synthetic likelihood
0.0
0.2
0.4
0.6
0.8
1.0
β index points


Final Sythetic Likelihood for γ_L Slice
Synthetic likelihood
0.00014
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
γ_L index points


Final Sythetic Likelihood for λ Slice
1e−5
Synthetic likelihood
2.5
2.0
1.5
1.0
0.5
0.0
0.00
0.02
0.04
0.06
0.08
0.10
λ index points






```
# print(index_vals[-600,].round(3))
# print(index_vals[-400,].round(3))
print(index_vals[-200,].round(3))
```

```
print(index_vals[-80,].round(3))
print(index_vals[-40,].round(3))
print(index_vals[-20,].round(3))
print(index_vals[-8,].round(3))
print(index_vals[-4,].round(3))
print(index_vals[-2,].round(3))
print(index_vals[-1,].round(3))
```

```
[0.137 0.129 0.003 0.033 0.039 0.026]
[0.785 0.041 0.033 0.058 0.071 0.011]
[0.582 0.7   0.001 0.043 0.021 0.016]
[0.701 0.265 0.031 0.02  0.017 0.026]
[0.311 0.062 0.031 0.041 0.022 0.021]
[0.7   0.268 0.013 0.05  0.017 0.026]
[0.7   0.268 0.027 0.05  0.017 0.026]
[0.7   0.268 0.033 0.05  0.017 0.026]
```