Inference on the Champagne Model using a Gaussian Process

TODO

- Set seed for LHC and stuff
- Change to log discrepency with custom observation variance
- Change from MLE to cross validation

Setting up the Champagne Model

Imports

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

from scipy.stats import qmc

import tensorflow as tf
import tensorflow_probability as tfp

tfb = tfp.bijectors
tfd = tfp.distributions
tfk = tfp.math.psd_kernels
```

```
2024-04-17 12:31:48.524588: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensorl
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-04-17 12:31:49.285584: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Wa
```

Model itself

```
np.random.seed(590154)

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

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


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

    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)

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

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

        total_rate = (
            S_0_to_I_L
            + S_0_to_S_L
            + I_0_to_S_0
            + I_0_to_I_L
            + I_L_to_I_0
            + I_L_to_S_L
            + S_L_to_S_0
            + S_L_to_I_L
        )

        delta_t = np.random.exponential(1 / total_rate)
```

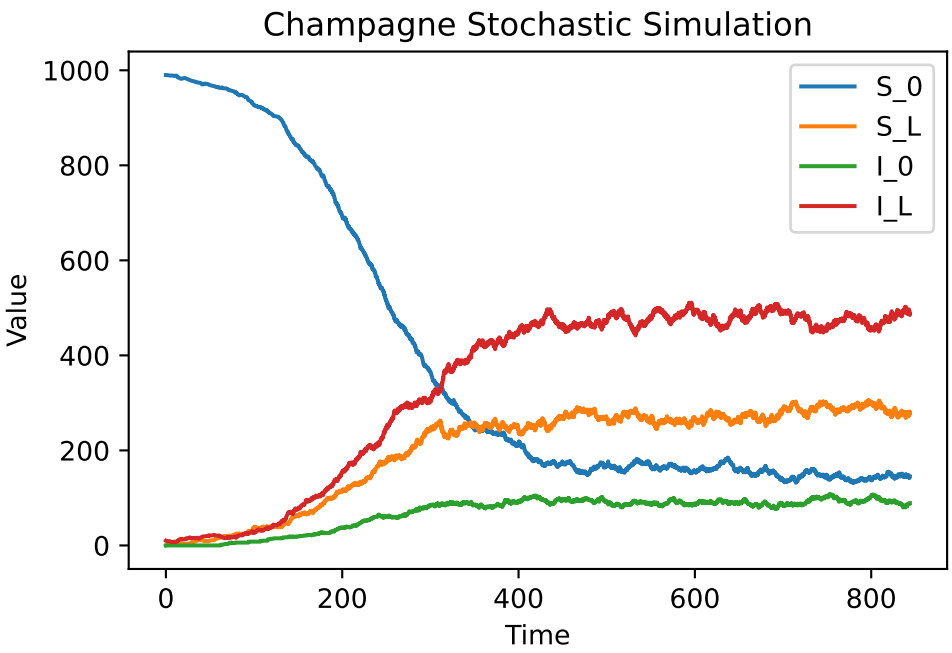
```
        new_stages_prob = [
            S_0_to_I_L / total_rate,
            S_0_to_S_L / total_rate,
            I_0_to_S_0 / total_rate,
            I_0_to_I_L / total_rate,
            I_L_to_I_0 / total_rate,
            I_L_to_S_L / total_rate,
            S_L_to_S_0 / total_rate,
         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

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

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

```
            p=new_stages_prob,
        )

        list_of_outcomes.append(new_stages)

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

    outcome_df = pd.DataFrame(list_of_outcomes)
    return outcome_df


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

Plotting outcome

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



Function that Outputs Final Prevalence

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


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

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


observed_sum_stats = champ_sum_stats(
```

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


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

Testing the variances across different values of params etc.

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

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

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

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

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

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

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

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

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

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

Gaussian Process Regression on Final Prevalence Discrepency

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

num_samples = 30

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

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

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

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

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

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

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

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

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

Generate Discrepencies

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

print(random_discrepencies.head())
```

```
0    0.542551
1    0.627749
2    0.650314
3    0.644435
4    0.667979
dtype: float64
```

Differing Methods to Iterate Function

```
# import timeit

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

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

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

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

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

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

Constrain Variables to be Positive

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

```
2024-04-17 12:32:24.462440: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:9
2024-04-17 12:32:24.603884: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot
Skipping registering GPU devices...
```

Custom Quadratic Mean Function

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

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

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

Making the ARD Kernel

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

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

```
observation_noise_variance_champ = tfp.util.TransformedVariable(
    bijector=constrain_positive,
    initial_value=0.03,
    dtype=np.float64,
    name="observation_noise_variance_champ",
)
```

```
length_scales_champ = tfp.util.TransformedVariable(
    bijector=constrain_positive,
    initial_value=[0.05, 0.05, 0.05, 0.05, 0.05, 0.05],
    dtype=np.float64,
    name="length_scales_champ",
)
```

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

Define the Gaussian Process with Quadratic Mean Function and ARD Kernel

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

print(champ_GP.trainable_variables)

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

```
(<tf.Variable 'amplitude_champ:0' shape=() dtype=float64, numpy=0.0>, <tf.Variable 'length_s
array([-2.99573227, -2.99573227, -2.99573227, -2.99573227, -2.99573227,
       -2.99573227])>, <tf.Variable 'observation_noise_variance_champ:0' shape=() dtype=float
```

Train the Hyperparameters

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


num_iters = 10000

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

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

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

    previous_loss = loss
```

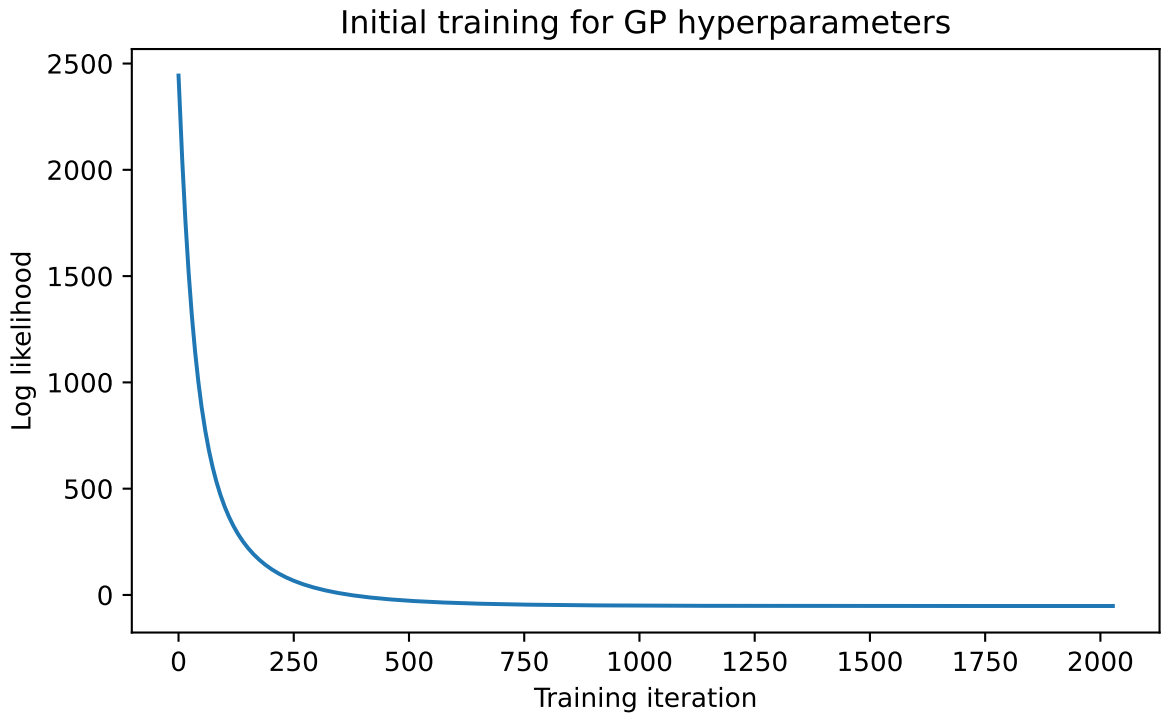
```
Hyperparameter convergence reached at iteration 2028.
```

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

```
Trained parameters:
amplitude_champ:0 is 0.812

length_scales_champ:0 is [0.004 0.014 0.032 0.017 0.022 0.018]

observation_noise_variance_champ:0 is 0.238

alpha_tp:0 is -0.575

amp_alpha_mean:0 is 0.252

amp_beta_mean:0 is 1.231

amp_f_mean:0 is 1128.319

amp_gamma_L_mean:0 is 4.401

amp_lambda_mean:0 is 96.25

amp_r_mean:0 is 48.536

beta_tp:0 is 0.522

bias_mean:0 is -1.332

f_tp:0 is 0.015

gamma_L_tp:0 is -0.103

lambda_tp:0 is 0.041
```

```
r_tp:0 is 0.173
```

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



Creating slices across one variable dimension

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

```
slice_samples_dict = {
    "alpha_slice_samples": np.repeat(np.concatenate(
```

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

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

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

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

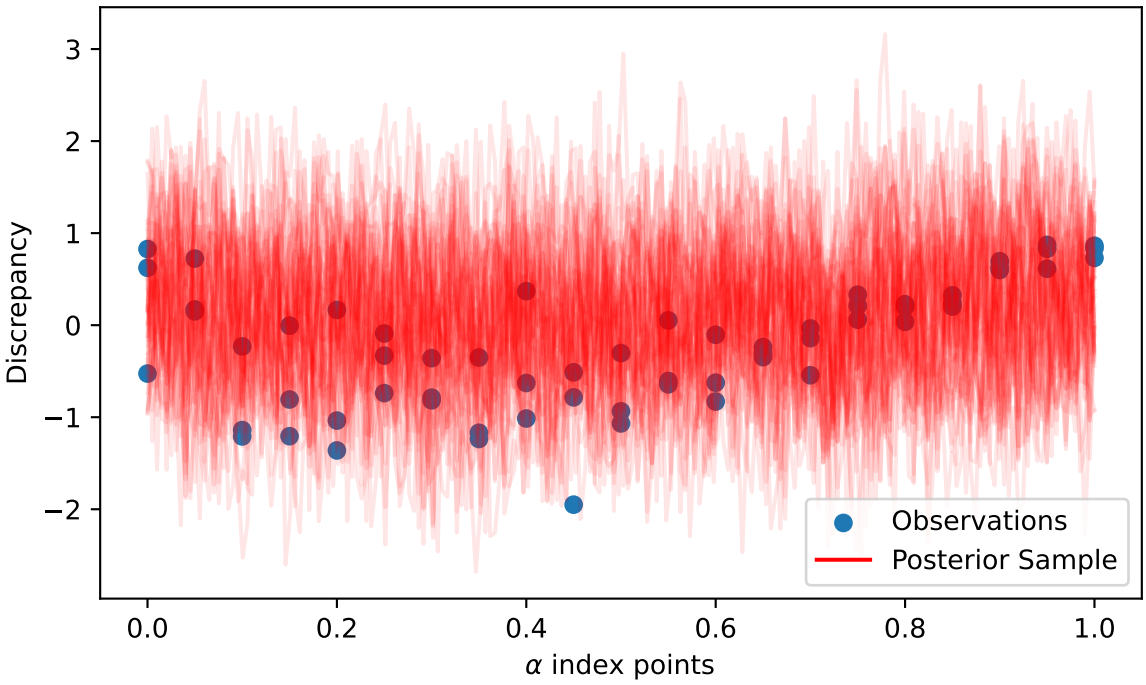
```
        (
            np.repeat(pv_champ_alpha, plot_gp_no).reshape(-1, 1),  # alpha
            np.repeat(pv_champ_beta, plot_gp_no).reshape(-1, 1),  # beta
            np.repeat(pv_champ_gamma_L, plot_gp_no).reshape(-1, 1),  # gamma_L
            np.repeat(pv_champ_lambda, plot_gp_no).reshape(-1,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

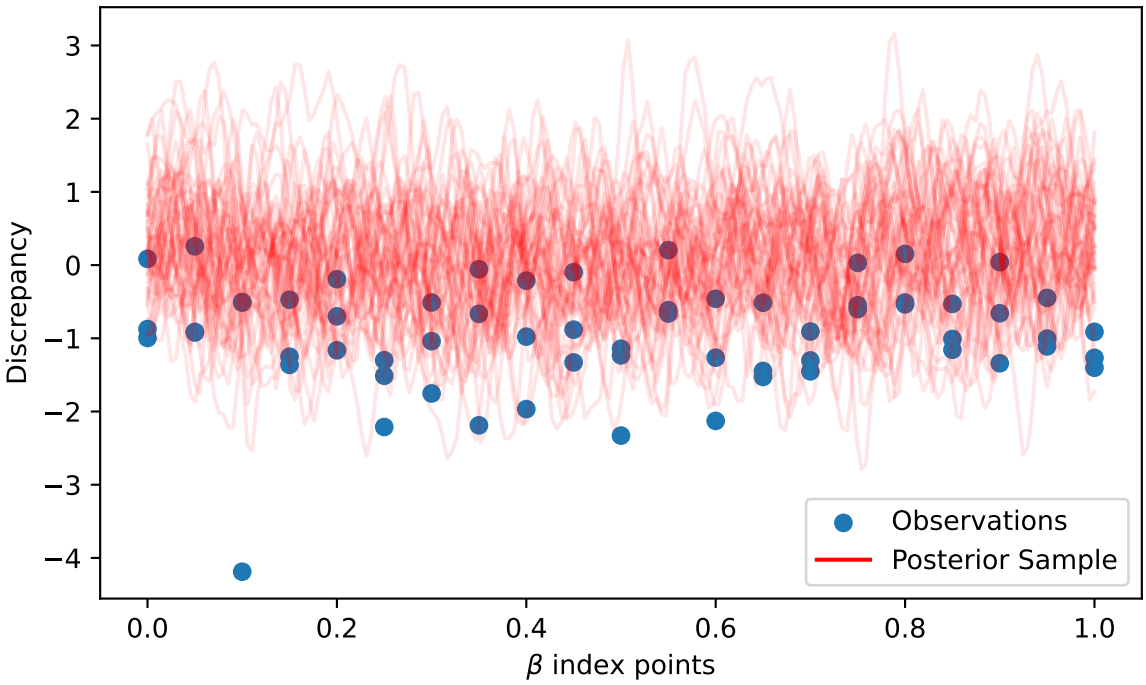
Plotting the GPs across different slices

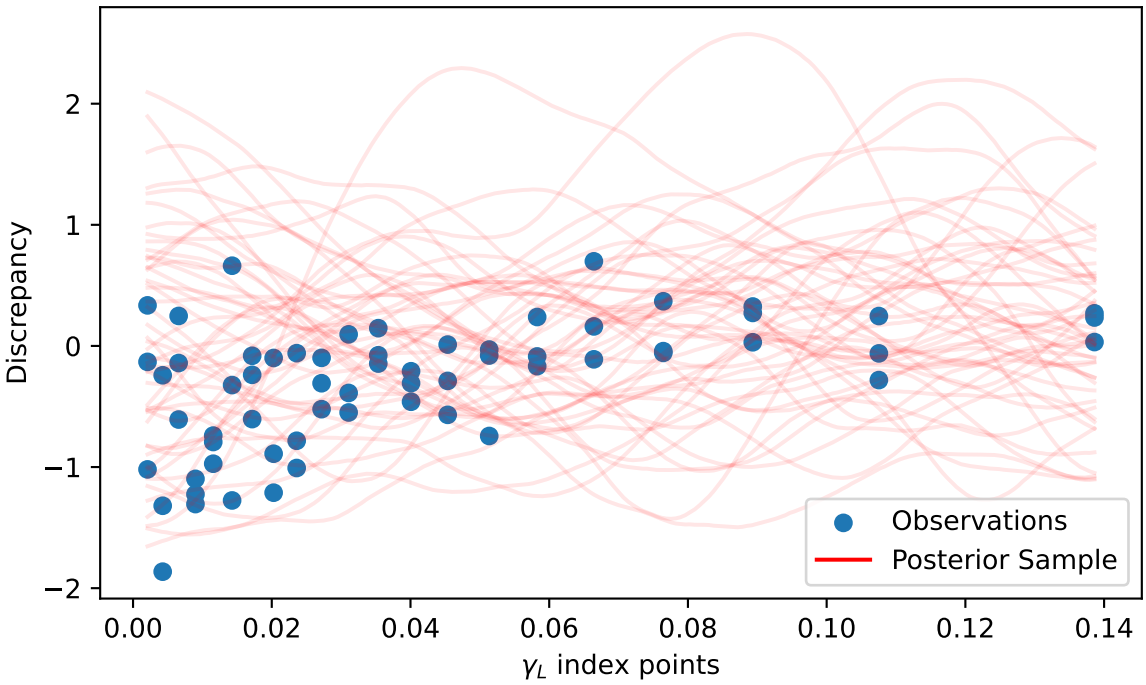
```
GP_seed = tfp.random.sanitize_seed(4362)
vars = ["alpha", "beta", "gamma_L", "lambda", "f", "r"]
slice_indices_dfs_dict = {}
slice_index_vals_dict = {}
slice_discrepencies_dict = {}

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

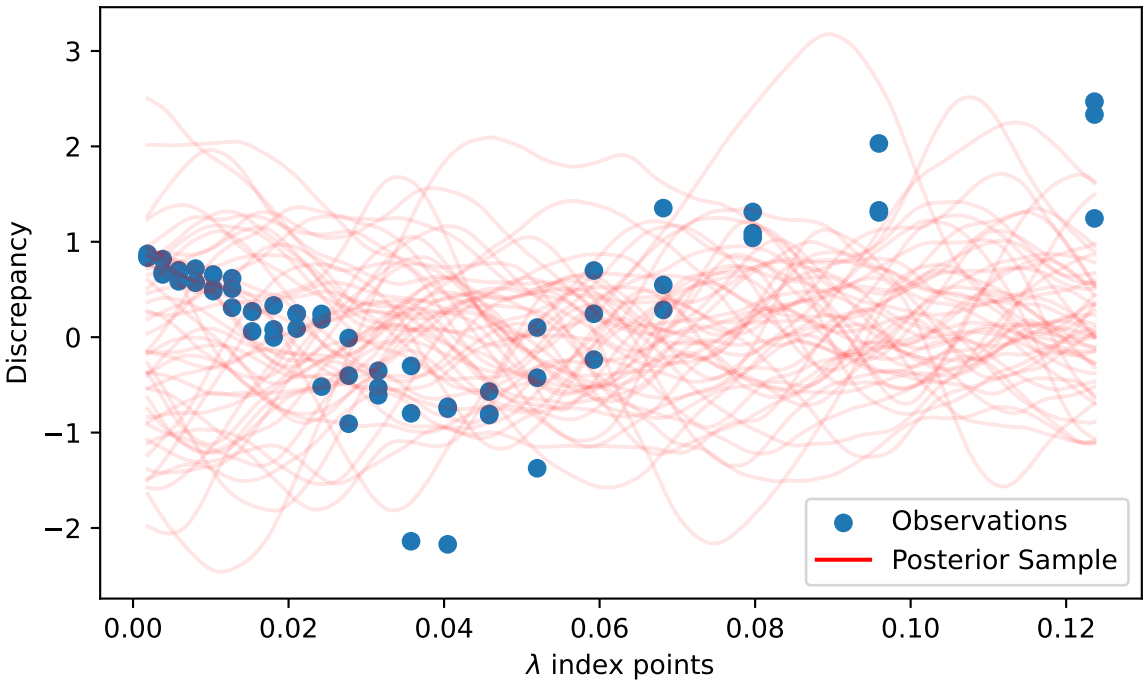
```
    )
    slice_discrepencies_dict[var + "_slice_discrepencies"] = discreps


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

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

    plt.figure(figsize=(7, 4))
    plt.scatter(
        val_df[var].values,
        discreps,
        label="Observations",
    )
    for i in range(gp_samp_no):
        plt.plot(
            gp_samples_df[var].values,
            GP_samples[i, :],
            c="r",
            alpha=0.1,
            label="Posterior Sample" if i == 0 else None,
        )
    leg = plt.legend(loc="lower right")
    for lh in leg.legend_handles:
        lh.set_alpha(1)
    if var in ["f", "r"]:
        plt.xlabel("$" + var + "$ index points")
    else:
        plt.xlabel("$\\" + var + "$ index points")
```

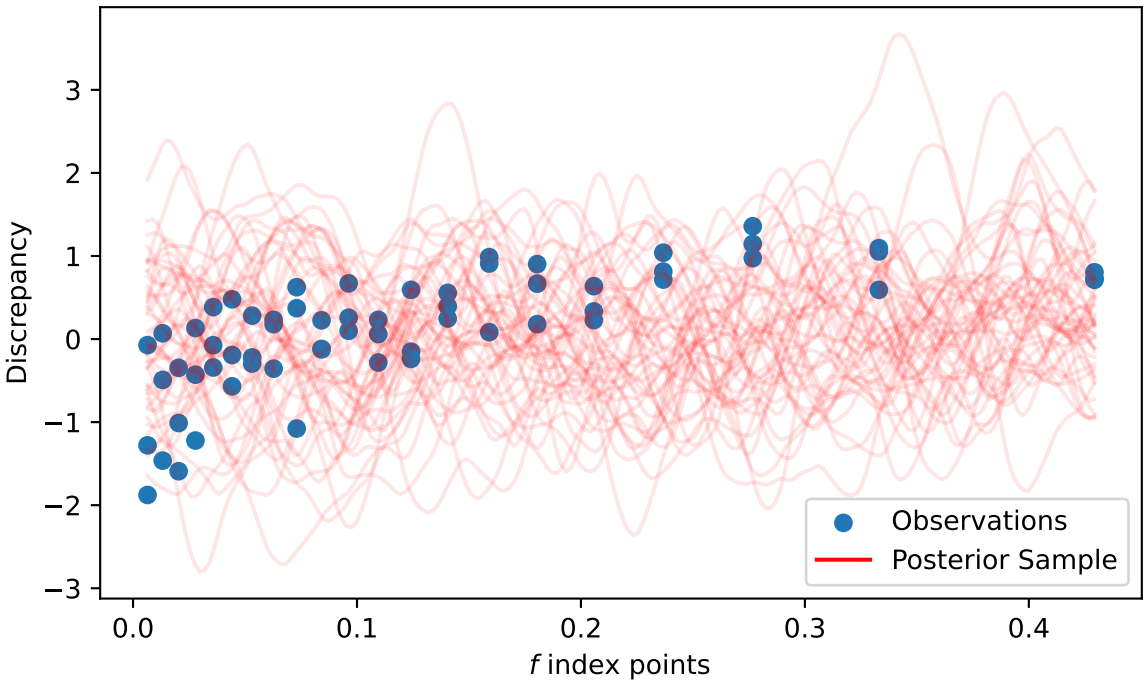
```
    # if var not in ["alpha", "beta"]:
    #     plt.xscale("log", base=np.e)
    plt.ylabel("Discrepancy")
    plt.savefig("champagne_GP_images/initial_" + var + "_slice.pdf")
    plt.show()
```

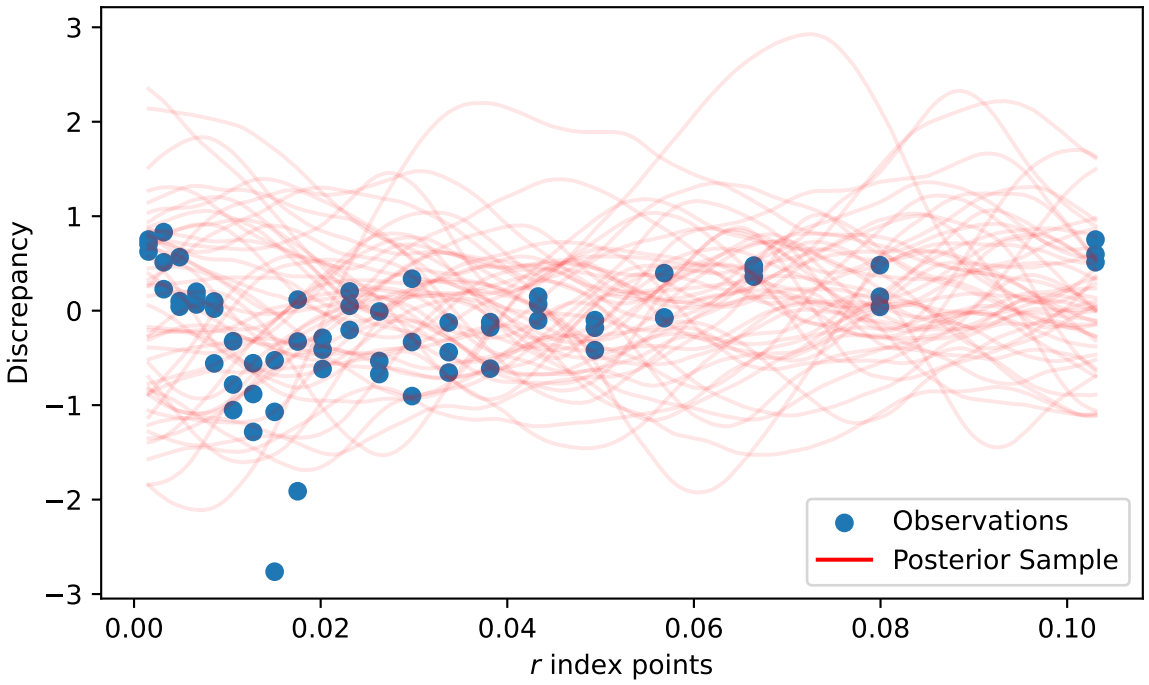



Discrepancy
3
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample


Discrepancy
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γ_L index points
Observations
Posterior Sample


Discrepancy
3
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample


Discrepancy
3
2
1
0
−1
−2
−3
0.0
0.1
0.2
0.3
0.4
f index points
Observations
Posterior Sample



Acquiring the next datapoint to test

Proof that .variance returns what we need in acquisition function

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

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

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

```
Self Kernel is 0.659
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. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[ 2.94799119e+000 -1.24829656e+000 -1.24829656e+000 ... -6.13992477e-146
  -6.13992477e-146 -6.13992477e-146]
 [-1.24829656e+000  2.94799119e+000 -1.24829656e+000 ... -6.13992477e-146
  -6.13992477e-146 -6.13992477e-146]
 [-1.24829656e+000 -1.24829656e+000  2.94799119e+000 ... -6.13992477e-146
  -6.13992477e-146 -6.13992477e-146]
 ...
 [-6.13992477e-146 -6.13992477e-146 -6.13992477e-146 ...  2.94799119e+000
  -1.24829656e+000 -1.24829656e+000]
 [-6.13992477e-146 -6.13992477e-146 -6.13992477e-146 ... -1.24829656e+000
   2.94799119e+000 -1.24829656e+000]
 [-6.13992477e-146 -6.13992477e-146 -6.13992477e-146 ... -1.24829656e+000
  -1.24829656e+000  2.94799119e+000]]
Variance function is [0.659]
Variance function is 0.659
```

Loss function

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

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

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

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

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

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

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

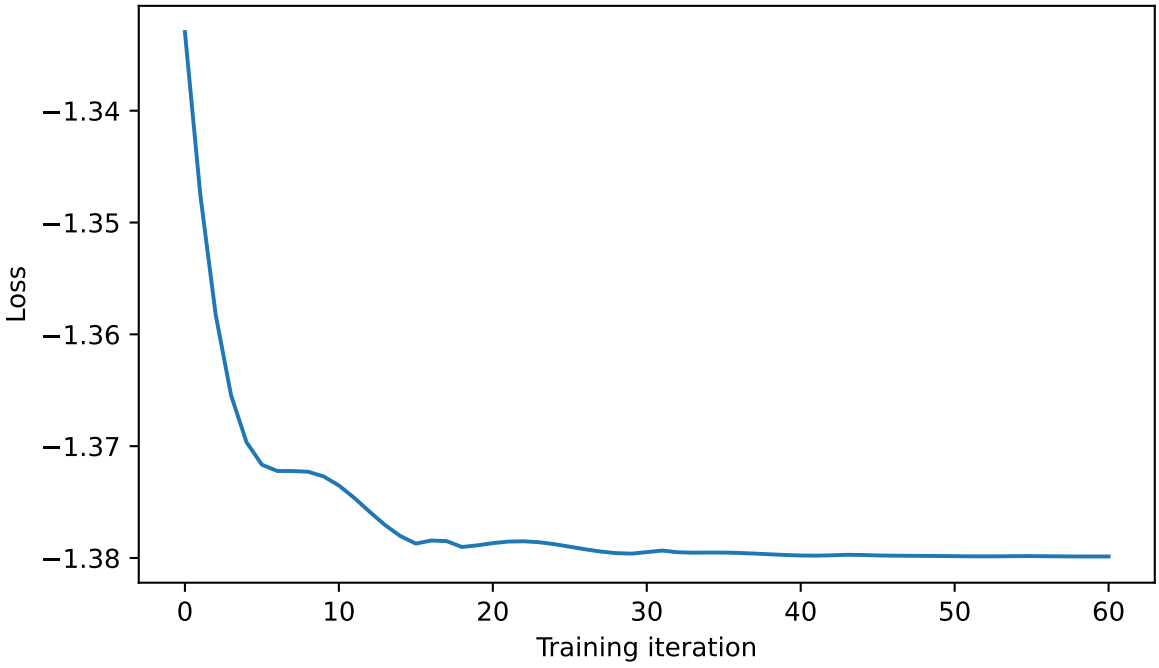
```
Adam_optim = tf.optimizers.Adam(learning_rate=0.1)


@tf.function(autograph=False, jit_compile=False)
def optimize():
    with tf.GradientTape() as tape:
        next_guess = tf.reshape(
            [
                tfb.Sigmoid().forward(next_vars[0]),
```

```
                tfb.Sigmoid().forward(next_vars[1]),
                constrain_positive.forward(next_vars[2]),
                constrain_positive.forward(next_vars[3]),
                constrain_positive.forward(next_vars[4]),
                constrain_positive.forward(next_vars[5]),
            ],
            [1, 6],
        )
        mean_t = champ_GP_reg.mean_fn(next_guess)
        std_t = champ_GP_reg.stddev(index_points=next_guess)
        loss = tf.squeeze(mean_t - 1.7 * std_t)
    grads = tape.gradient(loss, next_vars)
    Adam_optim.apply_gradients(zip(grads, next_vars))
    return loss


num_iters = 10000

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

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

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

    previous_loss = loss

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

```
            "{} is {}".format(
                var.name, constrain_positive.forward(var).numpy().round(3)
            )
        )
```

```
Acquisition function convergence reached at iteration 61.
Trained parameters:
next_alpha:0 is 0.39
next_beta:0 is 0.4
next_gamma_L:0 is 0.012
next_lambda:0 is 0.042
next_f:0 is 0.015
next_r:0 is 0.017
```

```
plt.figure(figsize=(7, 4))
plt.plot(lls_)
plt.xlabel("Training iteration")
plt.ylabel("Loss")
plt.savefig("champagne_GP_images/bolfi_optim_loss.pdf")
plt.show()
```



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

    num_iters = 10000

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

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

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

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


def update_var():
```

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

    num_iters = 10000

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

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

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

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

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


exploration_rate = 0.1
d = 6
update_freq = 20  # how many iterations before updating GP hyperparams

for t in range(201):
    optimizer_fast = tf.optimizers.Adam(learning_rate=0.01)
    optimizer_slow = tf.optimizers.Adam()
    eta_t = new_eta_t(t, d, exploration_rate)
    print(t)

    for var in next_vars:
        var.assign(0)
    update_var()

    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,
        np.array(
            [
                next_alpha.numpy(),
                next_beta.numpy(),
                next_gamma_L.numpy(),
                next_lambda.numpy(),
                next_f.numpy(),
                next_r.numpy(),
            ]
        ).reshape(1, -1),
        axis=0,
    )
    obs_vals = np.append(obs_vals, new_discrepency)
```

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

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

    if (t > 0) & (t % 50 == 0):
        print("Trained parameters:")
        for var in champ_GP.trainable_variables:
            if "tp" in var.name or "bias" in var.name:
                print("{} is {}\n".format(var.name, var.numpy().round(3)))
            else:
                print(
                    "{} is {}\n".format(
                        var.name, constrain_positive.forward(var).numpy().round(3)
                    )
                )
        for var in vars:
            champ_GP_reg = tfd.GaussianProcessRegressionModel(
                kernel=kernel_champ,
                index_points=slice_indices_dfs_dict[var + "_gp_indices_df"].values,
                observation_index_points=index_vals,
                observations=obs_vals,
                observation_noise_variance=observation_noise_variance_champ,
                predictive_noise_variance=0.0,
                mean_fn=quad_mean_fn(),
            )
            GP_samples = champ_GP_reg.sample(gp_samp_no, seed=GP_seed)

            plt.figure(figsize=(7, 4))
```

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

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

```
0
Acquisition function convergence reached at iteration 2291.
tf.Tensor(-1.5163768827573196, shape=(), dtype=float64)
Hyperparameter convergence reached at iteration 7270.
1
```

```
Acquisition function convergence reached at iteration 900.
tf.Tensor(-2.5466079698208017, shape=(), dtype=float64)
2
Acquisition function convergence reached at iteration 964.
tf.Tensor(-2.8658736359158414, shape=(), dtype=float64)
3
Acquisition function convergence reached at iteration 893.
tf.Tensor(-2.9395465936390437, shape=(), dtype=float64)
4
Acquisition function convergence reached at iteration 984.
tf.Tensor(-3.215264967552473, shape=(), dtype=float64)
5
Acquisition function convergence reached at iteration 2245.
tf.Tensor(-3.214185569507855, shape=(), dtype=float64)
6
Acquisition function convergence reached at iteration 2270.
tf.Tensor(-3.3122727186688006, shape=(), dtype=float64)
7
Acquisition function convergence reached at iteration 935.
tf.Tensor(-3.4596117925563448, shape=(), dtype=float64)
8
Acquisition function convergence reached at iteration 2056.
tf.Tensor(-3.5392157955444543, shape=(), dtype=float64)
9
Acquisition function convergence reached at iteration 2366.
tf.Tensor(-3.6905434184678887, shape=(), dtype=float64)
10
Acquisition function convergence reached at iteration 1090.
tf.Tensor(-3.7489805754081917, shape=(), dtype=float64)
11
Acquisition function convergence reached at iteration 2157.
tf.Tensor(-3.732044397735831, shape=(), dtype=float64)
12
Acquisition function convergence reached at iteration 1348.
tf.Tensor(-3.7604427583721116, shape=(), dtype=float64)
13
Acquisition function convergence reached at iteration 1146.
tf.Tensor(-3.8584942046685167, shape=(), dtype=float64)
14
Acquisition function convergence reached at iteration 1680.
tf.Tensor(-3.946568825891033, shape=(), dtype=float64)
15
Acquisition function convergence reached at iteration 819.
```

```
tf.Tensor(-3.7865908177627663, shape=(), dtype=float64)
16
Acquisition function convergence reached at iteration 1196.
tf.Tensor(-3.9432981062490824, shape=(), dtype=float64)
17
Acquisition function convergence reached at iteration 1629.
tf.Tensor(-3.9306338138795707, shape=(), dtype=float64)
18
Acquisition function convergence reached at iteration 1084.
tf.Tensor(-3.940713801971577, shape=(), dtype=float64)
19
Acquisition function convergence reached at iteration 1400.
tf.Tensor(-3.913965028233932, shape=(), dtype=float64)
20
Acquisition function convergence reached at iteration 1682.
tf.Tensor(-3.9392647733628383, shape=(), dtype=float64)
Hyperparameter convergence reached at iteration 6172.
21
Acquisition function convergence reached at iteration 2351.
tf.Tensor(-3.3118748428975593, shape=(), dtype=float64)
22
Acquisition function convergence reached at iteration 1092.
tf.Tensor(-3.3246611748003403, shape=(), dtype=float64)
23
Acquisition function convergence reached at iteration 2456.
tf.Tensor(-3.3497843004842807, shape=(), dtype=float64)
24
Acquisition function convergence reached at iteration 1329.
tf.Tensor(-3.360966120814422, shape=(), dtype=float64)
25
Acquisition function convergence reached at iteration 1052.
tf.Tensor(-3.372520989104028, shape=(), dtype=float64)
26
Acquisition function convergence reached at iteration 3151.
tf.Tensor(-3.39869269669452, shape=(), dtype=float64)
27
Acquisition function convergence reached at iteration 496.
tf.Tensor(-3.390088389758162, shape=(), dtype=float64)
28
Acquisition function convergence reached at iteration 2331.
tf.Tensor(-3.432072570817609, shape=(), dtype=float64)
29
Acquisition function convergence reached at iteration 881.
```

```
tf.Tensor(-3.6874494901085444, shape=(), dtype=float64)
30
Acquisition function convergence reached at iteration 883.
tf.Tensor(-3.699943003870999, shape=(), dtype=float64)
31
Acquisition function convergence reached at iteration 2381.
tf.Tensor(-3.4715739949677693, shape=(), dtype=float64)
32
Acquisition function convergence reached at iteration 1711.
tf.Tensor(-3.479070560455156, shape=(), dtype=float64)
33
Acquisition function convergence reached at iteration 753.
tf.Tensor(-3.479279339786197, shape=(), dtype=float64)
34
Acquisition function convergence reached at iteration 2441.
tf.Tensor(-3.502983933473586, shape=(), dtype=float64)
35
Acquisition function convergence reached at iteration 1077.
tf.Tensor(-3.5005179625840315, shape=(), dtype=float64)
36
Acquisition function convergence reached at iteration 2332.
tf.Tensor(-3.534121072876641, shape=(), dtype=float64)
37
Acquisition function convergence reached at iteration 2168.
tf.Tensor(-3.693871361463891, shape=(), dtype=float64)
38
Acquisition function convergence reached at iteration 887.
tf.Tensor(-3.7583258743829, shape=(), dtype=float64)
39
Acquisition function convergence reached at iteration 912.
tf.Tensor(-3.8093535413246244, shape=(), dtype=float64)
40
Acquisition function convergence reached at iteration 791.
tf.Tensor(-3.562488350021937, shape=(), dtype=float64)
41
Acquisition function convergence reached at iteration 743.
tf.Tensor(-4.420457390855951, shape=(), dtype=float64)
42
Acquisition function convergence reached at iteration 1664.
tf.Tensor(-4.356412327833556, shape=(), dtype=float64)
43
Acquisition function convergence reached at iteration 2757.
tf.Tensor(-4.265120418736622, shape=(), dtype=float64)
```

```
44
Acquisition function convergence reached at iteration 3040.
tf.Tensor(-4.417146130947978, shape=(), dtype=float64)
45
Acquisition function convergence reached at iteration 2445.
tf.Tensor(-4.29014947610263, shape=(), dtype=float64)
46
Acquisition function convergence reached at iteration 1306.
tf.Tensor(-4.308127694632794, shape=(), dtype=float64)
47
Acquisition function convergence reached at iteration 2161.
tf.Tensor(-4.305107991646819, shape=(), dtype=float64)
48
Acquisition function convergence reached at iteration 1340.
tf.Tensor(-4.436007310942894, shape=(), dtype=float64)
49
Acquisition function convergence reached at iteration 1310.
tf.Tensor(-4.351953598010001, shape=(), dtype=float64)
50
Acquisition function convergence reached at iteration 1439.
tf.Tensor(-4.3268745740490715, shape=(), dtype=float64)
Trained parameters:
amplitude_champ:0 is 0.902

length_scales_champ:0 is [2.000e-03 6.619e+00 1.011e+00 1.200e-02 2.324e+00 1.800e-02]

observation_noise_variance_champ:0 is 0.233

alpha_tp:0 is -0.364

amp_alpha_mean:0 is 0.213

amp_beta_mean:0 is 1.017

amp_f_mean:0 is 30.916

amp_gamma_L_mean:0 is 16.792

amp_lambda_mean:0 is 58.255

amp_r_mean:0 is 71.33

beta_tp:0 is 0.471
```

```
bias_mean:0 is -2.278

f_tp:0 is -0.158

gamma_L_tp:0 is -0.136

lambda_tp:0 is -0.025

r_tp:0 is 0.141

51
Acquisition function convergence reached at iteration 1577.
tf.Tensor(-4.3500817994321, shape=(), dtype=float64)
52
Acquisition function convergence reached at iteration 1283.
tf.Tensor(-4.367279791979929, shape=(), dtype=float64)
53
Acquisition function convergence reached at iteration 1018.
tf.Tensor(-4.362849704924937, shape=(), dtype=float64)
54
Acquisition function convergence reached at iteration 1154.
tf.Tensor(-4.352633840945179, shape=(), dtype=float64)
55
Acquisition function convergence reached at iteration 1413.
tf.Tensor(-4.351047397709436, shape=(), dtype=float64)
56
Acquisition function convergence reached at iteration 2096.
tf.Tensor(-4.621864762259046, shape=(), dtype=float64)
57
Acquisition function convergence reached at iteration 2125.
tf.Tensor(-4.4515264436439885, shape=(), dtype=float64)
58
Acquisition function convergence reached at iteration 2064.
tf.Tensor(-4.432598615829885, shape=(), dtype=float64)
59
Acquisition function convergence reached at iteration 1070.
tf.Tensor(-4.431889403558616, shape=(), dtype=float64)
60
Acquisition function convergence reached at iteration 1862.
tf.Tensor(-4.412102506394968, shape=(), dtype=float64)
61
Acquisition function convergence reached at iteration 1127.
```

```
tf.Tensor(-3.9611990753365185, shape=(), dtype=float64)
62
Acquisition function convergence reached at iteration 2842.
tf.Tensor(-3.9816200483649475, shape=(), dtype=float64)
63
Acquisition function convergence reached at iteration 932.
tf.Tensor(-3.9648850841588237, shape=(), dtype=float64)
64
Acquisition function convergence reached at iteration 1001.
tf.Tensor(-3.9924604919421243, shape=(), dtype=float64)
65
Acquisition function convergence reached at iteration 2153.
tf.Tensor(-4.049037731294646, shape=(), dtype=float64)
66
Acquisition function convergence reached at iteration 1474.
tf.Tensor(-4.028916738803897, shape=(), dtype=float64)
67
Acquisition function convergence reached at iteration 1174.
tf.Tensor(-4.015599759389074, shape=(), dtype=float64)
68
Acquisition function convergence reached at iteration 2366.
tf.Tensor(-4.031484479660779, shape=(), dtype=float64)
69
Acquisition function convergence reached at iteration 2195.
tf.Tensor(-4.031505600666671, shape=(), dtype=float64)
70
Acquisition function convergence reached at iteration 1500.
tf.Tensor(-4.027152873369964, shape=(), dtype=float64)
71
Acquisition function convergence reached at iteration 1255.
tf.Tensor(-4.025276596398787, shape=(), dtype=float64)
72
Acquisition function convergence reached at iteration 1425.
tf.Tensor(-4.0158271612114556, shape=(), dtype=float64)
73
Acquisition function convergence reached at iteration 1007.
tf.Tensor(-4.010343046166209, shape=(), dtype=float64)
74
Acquisition function convergence reached at iteration 1318.
tf.Tensor(-4.057390761421983, shape=(), dtype=float64)
75
Acquisition function convergence reached at iteration 2029.
tf.Tensor(-4.0644950687002686, shape=(), dtype=float64)
```

```
76
Acquisition function convergence reached at iteration 1279.
tf.Tensor(-4.0628354126953, shape=(), dtype=float64)
77
Acquisition function convergence reached at iteration 956.
tf.Tensor(-4.072580442105861, shape=(), dtype=float64)
78
Acquisition function convergence reached at iteration 1838.
tf.Tensor(-4.074162506715589, shape=(), dtype=float64)
79
Acquisition function convergence reached at iteration 1126.
tf.Tensor(-4.070790539983008, shape=(), dtype=float64)
80
Acquisition function convergence reached at iteration 873.
tf.Tensor(-4.068387472902019, shape=(), dtype=float64)
Hyperparameter convergence reached at iteration 9223.
81
Acquisition function convergence reached at iteration 738.
tf.Tensor(-3.828677782122727, shape=(), dtype=float64)
82
Acquisition function convergence reached at iteration 901.
tf.Tensor(-3.829406904481869, shape=(), dtype=float64)
83
Acquisition function convergence reached at iteration 2071.
tf.Tensor(-3.857856813180537, shape=(), dtype=float64)
84
Acquisition function convergence reached at iteration 2124.
tf.Tensor(-3.8403220846986916, shape=(), dtype=float64)
85
Acquisition function convergence reached at iteration 2136.
tf.Tensor(-3.8445092576377915, shape=(), dtype=float64)
86
Acquisition function convergence reached at iteration 2139.
tf.Tensor(-3.848809911994335, shape=(), dtype=float64)
87
Acquisition function convergence reached at iteration 2141.
tf.Tensor(-3.853086963852988, shape=(), dtype=float64)
88
Acquisition function convergence reached at iteration 2143.
tf.Tensor(-3.8572962125386483, shape=(), dtype=float64)
89
Acquisition function convergence reached at iteration 2144.
tf.Tensor(-3.8614762898488566, shape=(), dtype=float64)
```

```
90
Acquisition function convergence reached at iteration 2144.
tf.Tensor(-3.8656215773093265, shape=(), dtype=float64)
91
Acquisition function convergence reached at iteration 2144.
tf.Tensor(-3.869705077277334, shape=(), dtype=float64)
92
Acquisition function convergence reached at iteration 2143.
tf.Tensor(-3.87376776025548, shape=(), dtype=float64)
93
Acquisition function convergence reached at iteration 2144.
tf.Tensor(-3.8777585349754697, shape=(), dtype=float64)
94
Acquisition function convergence reached at iteration 2144.
tf.Tensor(-3.8817017603877475, shape=(), dtype=float64)
95
Acquisition function convergence reached at iteration 2145.
tf.Tensor(-3.885597229149667, shape=(), dtype=float64)
96
Acquisition function convergence reached at iteration 2146.
tf.Tensor(-3.889447352964622, shape=(), dtype=float64)
97
Acquisition function convergence reached at iteration 2146.
tf.Tensor(-3.893264485408307, shape=(), dtype=float64)
98
Acquisition function convergence reached at iteration 2146.
tf.Tensor(-3.8970419138798675, shape=(), dtype=float64)
99
Acquisition function convergence reached at iteration 2146.
tf.Tensor(-3.900773020717868, shape=(), dtype=float64)
100
Acquisition function convergence reached at iteration 2146.
tf.Tensor(-3.904476828278494, shape=(), dtype=float64)
Hyperparameter convergence reached at iteration 9415.
Trained parameters:
amplitude_champ:0 is 0.745

length_scales_champ:0 is [2.00000e-03 1.62928e+03 6.70000e-02 1.10000e-02 1.98319e+02 1.7000

observation_noise_variance_champ:0 is 0.265

alpha_tp:0 is -0.158
```

```
amp_alpha_mean:0 is 0.37

amp_beta_mean:0 is 0.695

amp_f_mean:0 is 28.206

amp_gamma_L_mean:0 is 1.435

amp_lambda_mean:0 is 20.891

amp_r_mean:0 is 41.62

beta_tp:0 is 0.543

bias_mean:0 is -2.239

f_tp:0 is -0.105

gamma_L_tp:0 is 0.065

lambda_tp:0 is -0.241

r_tp:0 is 0.133

101
Acquisition function convergence reached at iteration 1021.
tf.Tensor(-3.797817652562625, shape=(), dtype=float64)
102
Acquisition function convergence reached at iteration 989.
tf.Tensor(-3.8010692919392737, shape=(), dtype=float64)
103
Acquisition function convergence reached at iteration 1769.
tf.Tensor(-3.8076156924941262, shape=(), dtype=float64)
104
Acquisition function convergence reached at iteration 1013.
tf.Tensor(-3.808191846347876, shape=(), dtype=float64)
105
Acquisition function convergence reached at iteration 1019.
tf.Tensor(-3.8116601419273715, shape=(), dtype=float64)
106
Acquisition function convergence reached at iteration 999.
tf.Tensor(-3.81485497134446, shape=(), dtype=float64)
107
```

```
Acquisition function convergence reached at iteration 1001.
tf.Tensor(-3.818209888192824, shape=(), dtype=float64)
108
Acquisition function convergence reached at iteration 1017.
tf.Tensor(-3.8216610738132304, shape=(), dtype=float64)
109
Acquisition function convergence reached at iteration 991.
tf.Tensor(-3.824724460483641, shape=(), dtype=float64)
110
Acquisition function convergence reached at iteration 1017.
tf.Tensor(-3.8281762735764047, shape=(), dtype=float64)
111
Acquisition function convergence reached at iteration 987.
tf.Tensor(-3.8311401280400053, shape=(), dtype=float64)
112
Acquisition function convergence reached at iteration 1011.
tf.Tensor(-3.8345136428436493, shape=(), dtype=float64)
113
Acquisition function convergence reached at iteration 999.
tf.Tensor(-3.837554099859927, shape=(), dtype=float64)
114
Acquisition function convergence reached at iteration 1010.
tf.Tensor(-3.8407777294660703, shape=(), dtype=float64)
115
Acquisition function convergence reached at iteration 1010.
tf.Tensor(-3.843860101827685, shape=(), dtype=float64)
116
Acquisition function convergence reached at iteration 997.
tf.Tensor(-3.8468034088213057, shape=(), dtype=float64)
117
Acquisition function convergence reached at iteration 1004.
tf.Tensor(-3.8499117745425777, shape=(), dtype=float64)
118
Acquisition function convergence reached at iteration 1011.
tf.Tensor(-3.8529742190981078, shape=(), dtype=float64)
119
Acquisition function convergence reached at iteration 982.
tf.Tensor(-3.855710617938112, shape=(), dtype=float64)
120
Acquisition function convergence reached at iteration 1876.
tf.Tensor(-3.8619539781633274, shape=(), dtype=float64)
121
Acquisition function convergence reached at iteration 916.
```

```
tf.Tensor(-3.7436034465340886, shape=(), dtype=float64)
122
Acquisition function convergence reached at iteration 404.
tf.Tensor(-3.716266946683087, shape=(), dtype=float64)
123
Acquisition function convergence reached at iteration 2357.
tf.Tensor(-3.7519895799255014, shape=(), dtype=float64)
124
Acquisition function convergence reached at iteration 1556.
tf.Tensor(-3.740676928932892, shape=(), dtype=float64)
125
Acquisition function convergence reached at iteration 1589.
tf.Tensor(-3.7432402757280623, shape=(), dtype=float64)
126
Acquisition function convergence reached at iteration 1560.
tf.Tensor(-3.7457220288910715, shape=(), dtype=float64)
127
Acquisition function convergence reached at iteration 1563.
tf.Tensor(-3.7484683223073816, shape=(), dtype=float64)
128
Acquisition function convergence reached at iteration 1561.
tf.Tensor(-3.751079917664207, shape=(), dtype=float64)
129
Acquisition function convergence reached at iteration 1558.
tf.Tensor(-3.7536849161969243, shape=(), dtype=float64)
130
Acquisition function convergence reached at iteration 1562.
tf.Tensor(-3.7562670348609144, shape=(), dtype=float64)
131
Acquisition function convergence reached at iteration 1557.
tf.Tensor(-3.758809203039427, shape=(), dtype=float64)
132
Acquisition function convergence reached at iteration 1559.
tf.Tensor(-3.7613676736447053, shape=(), dtype=float64)
133
Acquisition function convergence reached at iteration 1564.
tf.Tensor(-3.763947114460944, shape=(), dtype=float64)
134
Acquisition function convergence reached at iteration 1562.
tf.Tensor(-3.766455958008728, shape=(), dtype=float64)
135
Acquisition function convergence reached at iteration 1560.
tf.Tensor(-3.7689385346358164, shape=(), dtype=float64)
```

```
136
Acquisition function convergence reached at iteration 1554.
tf.Tensor(-3.771396612179421, shape=(), dtype=float64)
137
Acquisition function convergence reached at iteration 1555.
tf.Tensor(-3.77386894441014, shape=(), dtype=float64)
138
Acquisition function convergence reached at iteration 1557.
tf.Tensor(-3.7763088154293802, shape=(), dtype=float64)
139
Acquisition function convergence reached at iteration 1555.
tf.Tensor(-3.7787503896622883, shape=(), dtype=float64)
140
Acquisition function convergence reached at iteration 1559.
tf.Tensor(-3.7811624273605493, shape=(), dtype=float64)
Hyperparameter convergence reached at iteration 9019.
141
Acquisition function convergence reached at iteration 2055.
tf.Tensor(-3.672400713284797, shape=(), dtype=float64)
142
Acquisition function convergence reached at iteration 2055.
tf.Tensor(-3.6745438479284562, shape=(), dtype=float64)
143
Acquisition function convergence reached at iteration 1803.
tf.Tensor(-3.6770665921209393, shape=(), dtype=float64)
144
Acquisition function convergence reached at iteration 1961.
tf.Tensor(-3.673693688665503, shape=(), dtype=float64)
145
Acquisition function convergence reached at iteration 2199.
tf.Tensor(-3.690054421675976, shape=(), dtype=float64)
146
Acquisition function convergence reached at iteration 1816.
tf.Tensor(-3.677035103720063, shape=(), dtype=float64)
147
Acquisition function convergence reached at iteration 1540.
tf.Tensor(-3.6786475234902922, shape=(), dtype=float64)
148
Acquisition function convergence reached at iteration 1680.
tf.Tensor(-3.6811199569377613, shape=(), dtype=float64)
149
Acquisition function convergence reached at iteration 1387.
tf.Tensor(-3.682579130329113, shape=(), dtype=float64)
```

```
150
Acquisition function convergence reached at iteration 1546.
tf.Tensor(-3.6851646456083134, shape=(), dtype=float64)
Trained parameters:
amplitude_champ:0 is 0.698

length_scales_champ:0 is [2.000000e-03 2.517363e+03 4.720000e-01 1.000000e-02 8.482120e+02
 1.500000e-02]

observation_noise_variance_champ:0 is 0.329

alpha_tp:0 is -0.041

amp_alpha_mean:0 is 0.48

amp_beta_mean:0 is 0.642

amp_f_mean:0 is 12.327

amp_gamma_L_mean:0 is 1.858

amp_lambda_mean:0 is 20.62

amp_r_mean:0 is 27.223

beta_tp:0 is 0.557

bias_mean:0 is -1.942

f_tp:0 is -0.093

gamma_L_tp:0 is 0.176

lambda_tp:0 is -0.253

r_tp:0 is 0.13

151
Acquisition function convergence reached at iteration 1815.
tf.Tensor(-3.687804951201914, shape=(), dtype=float64)
152
Acquisition function convergence reached at iteration 1709.
tf.Tensor(-3.6897743954673725, shape=(), dtype=float64)
```

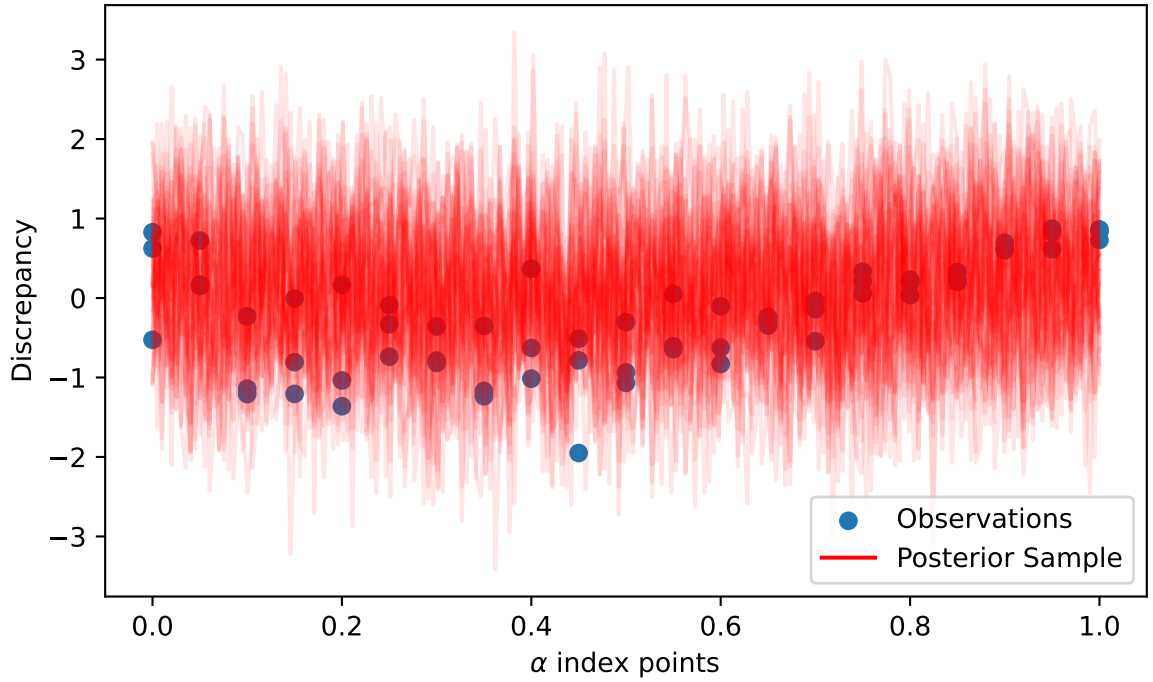
```
153
Acquisition function convergence reached at iteration 1810.
tf.Tensor(-3.692061630907383, shape=(), dtype=float64)
154
Acquisition function convergence reached at iteration 2032.
tf.Tensor(-3.694446461951503, shape=(), dtype=float64)
155
Acquisition function convergence reached at iteration 2056.
tf.Tensor(-3.6965650100850898, shape=(), dtype=float64)
156
Acquisition function convergence reached at iteration 1613.
tf.Tensor(-3.6979981353607654, shape=(), dtype=float64)
157
Acquisition function convergence reached at iteration 1416.
tf.Tensor(-3.6995675686689675, shape=(), dtype=float64)
158
Acquisition function convergence reached at iteration 1387.
tf.Tensor(-3.7015270563748826, shape=(), dtype=float64)
159
Acquisition function convergence reached at iteration 1391.
tf.Tensor(-3.7035739326508, shape=(), dtype=float64)
160
Acquisition function convergence reached at iteration 1444.
tf.Tensor(-3.7057549027955377, shape=(), dtype=float64)
161
Acquisition function convergence reached at iteration 1359.
tf.Tensor(-3.706564644440708, shape=(), dtype=float64)
162
Acquisition function convergence reached at iteration 674.
tf.Tensor(-3.7005248446422687, shape=(), dtype=float64)
163
Acquisition function convergence reached at iteration 2855.
tf.Tensor(-3.758755877040594, shape=(), dtype=float64)
164
Acquisition function convergence reached at iteration 2209.
tf.Tensor(-3.7501747994812464, shape=(), dtype=float64)
165
Acquisition function convergence reached at iteration 1375.
tf.Tensor(-3.7622159051917703, shape=(), dtype=float64)
166
Acquisition function convergence reached at iteration 1687.
tf.Tensor(-3.760754446849764, shape=(), dtype=float64)
167
```

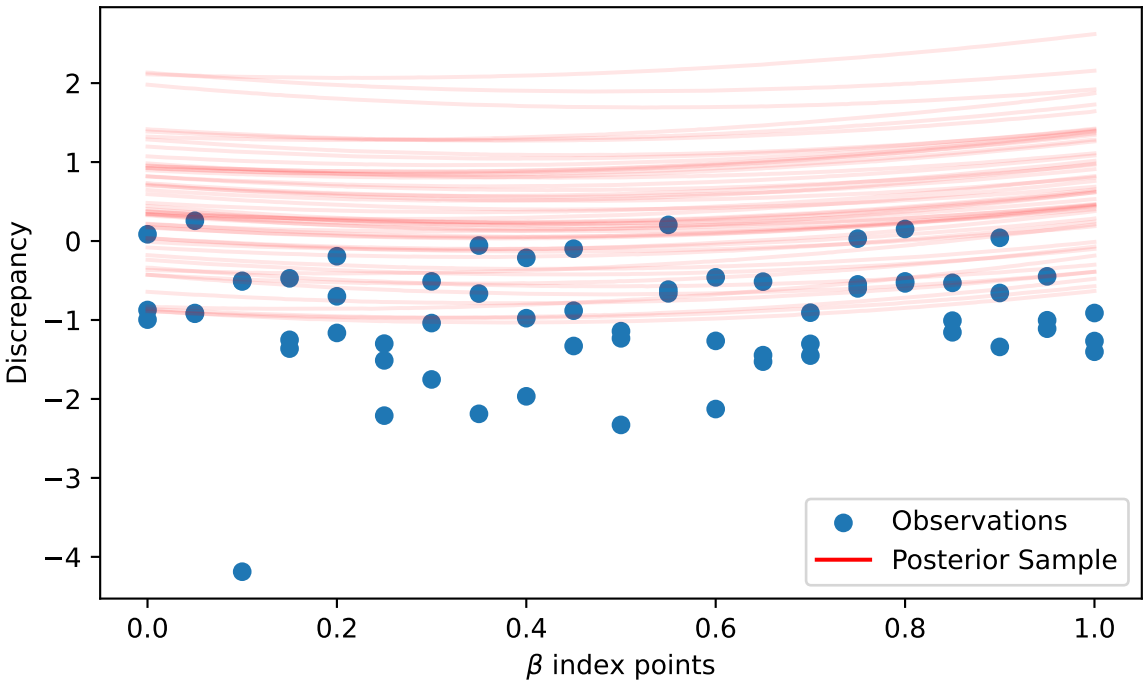
```
Acquisition function convergence reached at iteration 2400.
tf.Tensor(-3.755608147093165, shape=(), dtype=float64)
168
Acquisition function convergence reached at iteration 2147.
tf.Tensor(-3.7473093283544614, shape=(), dtype=float64)
169
Acquisition function convergence reached at iteration 2202.
tf.Tensor(-3.7880833546471155, shape=(), dtype=float64)
170
Acquisition function convergence reached at iteration 722.
tf.Tensor(-3.763428491087183, shape=(), dtype=float64)
171
Acquisition function convergence reached at iteration 2125.
tf.Tensor(-3.7647875490115297, shape=(), dtype=float64)
172
Acquisition function convergence reached at iteration 1229.
tf.Tensor(-3.7601403172214685, shape=(), dtype=float64)
173
Acquisition function convergence reached at iteration 769.
tf.Tensor(-3.7460719684260266, shape=(), dtype=float64)
174
Acquisition function convergence reached at iteration 1159.
tf.Tensor(-3.744324528850168, shape=(), dtype=float64)
175
Acquisition function convergence reached at iteration 1048.
tf.Tensor(-3.741402862422385, shape=(), dtype=float64)
176
Acquisition function convergence reached at iteration 934.
tf.Tensor(-3.7378771469251633, shape=(), dtype=float64)
177
Acquisition function convergence reached at iteration 911.
tf.Tensor(-3.7717171776907135, shape=(), dtype=float64)
178
Acquisition function convergence reached at iteration 1595.
tf.Tensor(-3.7668212921972346, shape=(), dtype=float64)
179
Acquisition function convergence reached at iteration 717.
tf.Tensor(-3.7472768092496733, shape=(), dtype=float64)
180
Acquisition function convergence reached at iteration 874.
tf.Tensor(-3.738046450567724, shape=(), dtype=float64)
181
Acquisition function convergence reached at iteration 2902.
```

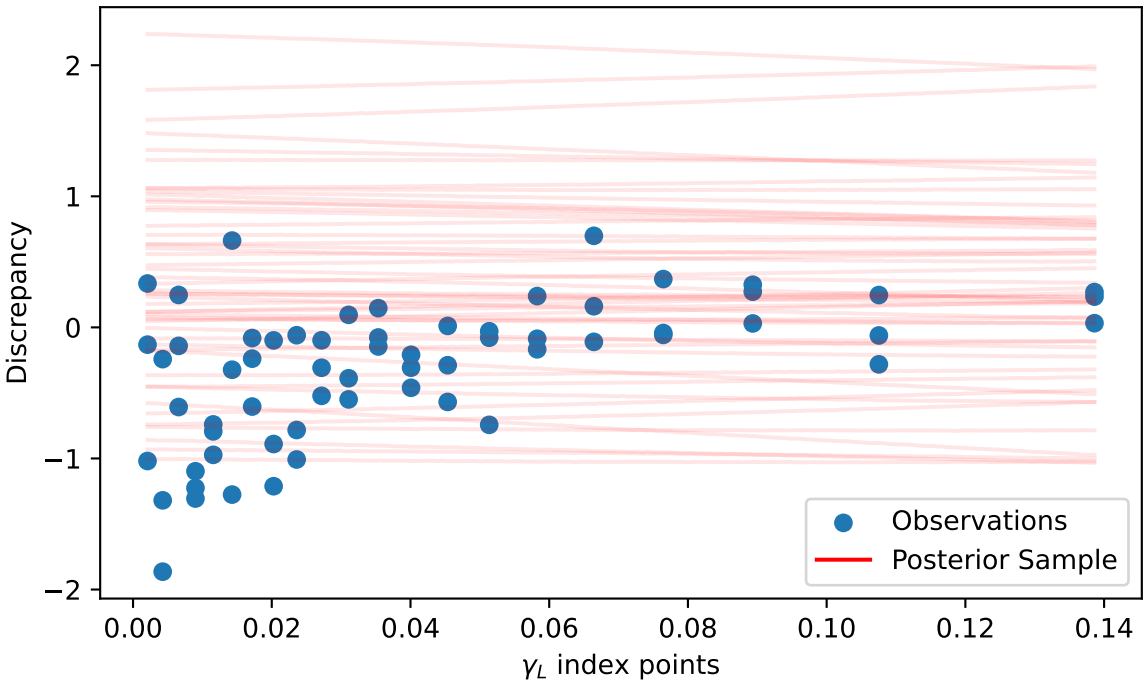
```
tf.Tensor(-3.7123878743784284, shape=(), dtype=float64)
182
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.71417343230627, shape=(), dtype=float64)
183
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.715948870909949, shape=(), dtype=float64)
184
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.717713966216459, shape=(), dtype=float64)
185
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7194687828573936, shape=(), dtype=float64)
186
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.721213414316947, shape=(), dtype=float64)
187
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.72294796987647, shape=(), dtype=float64)
188
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7246725545725634, shape=(), dtype=float64)
189
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.726387279788597, shape=(), dtype=float64)
190
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7280922518066677, shape=(), dtype=float64)
191
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7297875756325056, shape=(), dtype=float64)
192
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.73147335701008, shape=(), dtype=float64)
193
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7331496995785085, shape=(), dtype=float64)
194
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.734816702990741, shape=(), dtype=float64)
195
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7364744688559726, shape=(), dtype=float64)
```

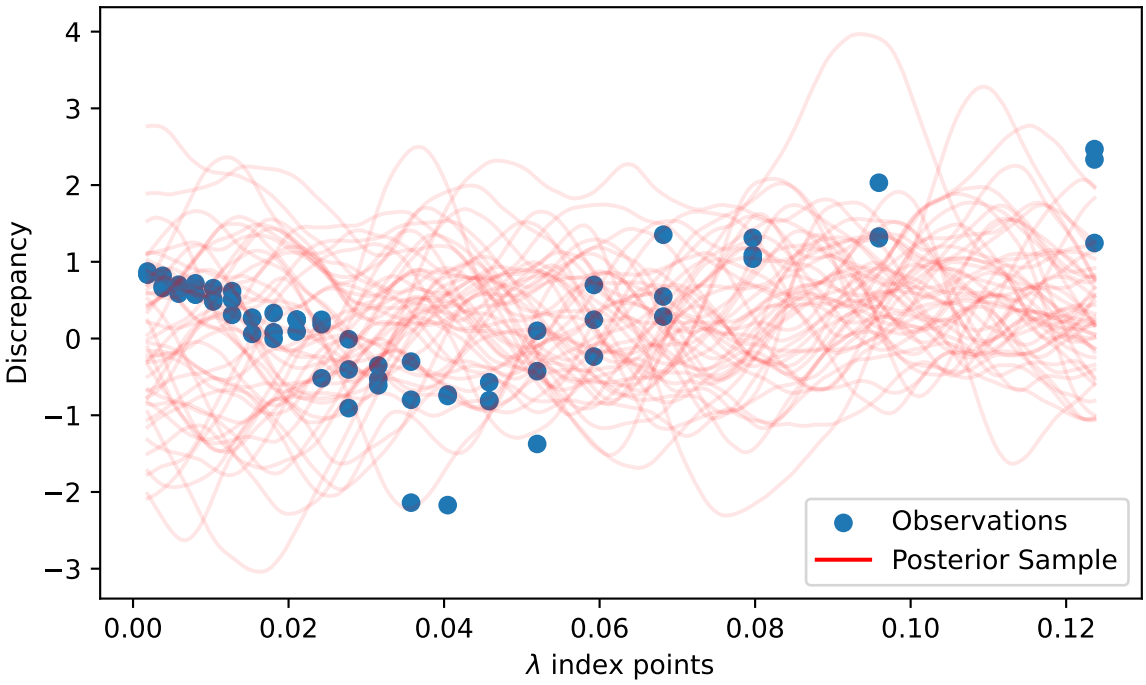
```
196
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.738123094869326, shape=(), dtype=float64)
197
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7397626780298454, shape=(), dtype=float64)
198
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.741393313364842, shape=(), dtype=float64)
199
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.7430150955764456, shape=(), dtype=float64)
200
Acquisition function convergence reached at iteration 2902.
tf.Tensor(-3.744628116494445, shape=(), dtype=float64)
Trained parameters:
amplitude_champ:0 is 0.887

length_scales_champ:0 is [1.5600000e-01 3.5671314e+04 3.0325000e+01 3.4000000e-02 4.3107910e-
 7.1000000e-02]

observation_noise_variance_champ:0 is 0.409

alpha_tp:0 is 3.065

amp_alpha_mean:0 is 0.139

amp_beta_mean:0 is 0.459

amp_f_mean:0 is 2.587

amp_gamma_L_mean:0 is 15.445

amp_lambda_mean:0 is 3.678

amp_r_mean:0 is 69.633

beta_tp:0 is 0.376

bias_mean:0 is -1.744

f_tp:0 is -0.356
```

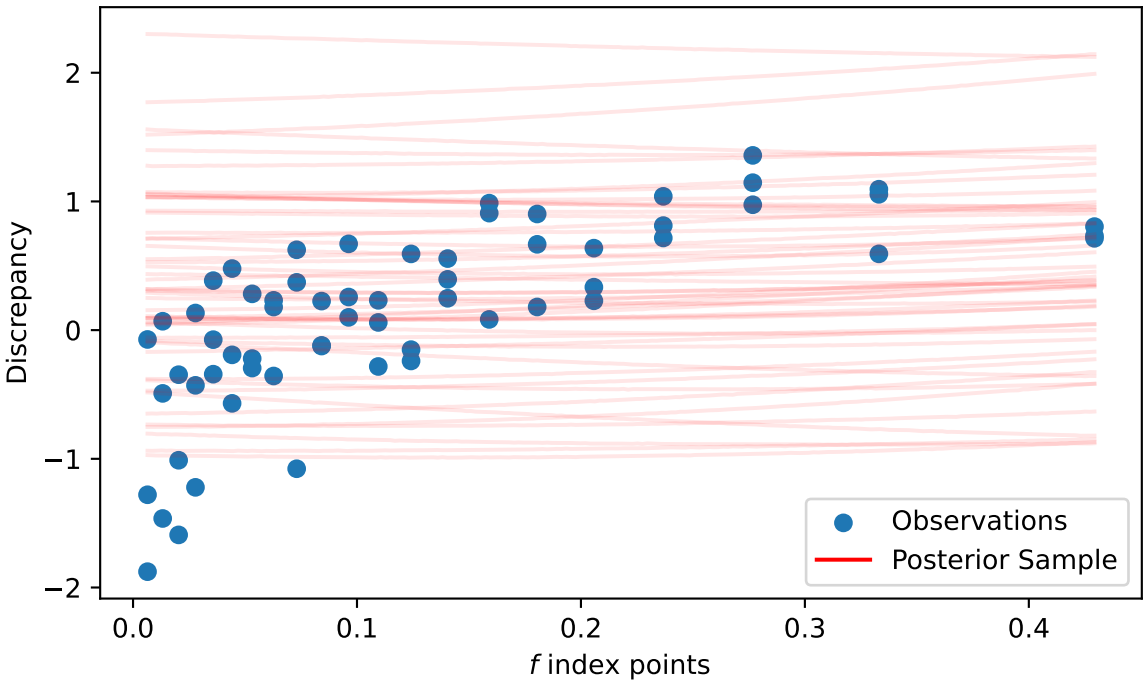
```
gamma_L_tp:0 is -0.008

lambda_tp:0 is -0.555

r_tp:0 is 0.124

[0.3925952  0.40608836 0.04250169 0.04438296 0.04092629 0.03262489]
[0.42984734 0.4        0.06401678 0.19098945 0.06296256 0.07732367]
[0.40531166 0.4        0.04235232 0.2746941  0.04516997 0.13894885]
[0.39863255 0.4        0.02017372 0.64845577 0.02421308 0.00936648]
[0.39863234 0.4        0.02017372 0.64845169 0.02421308 0.00936554]
[0.39863233 0.4        0.02017372 0.64845209 0.02421308 0.00936553]
[0.39863233 0.4        0.02017372 0.6484523  0.02421308 0.00936553]
[0.39863233 0.4        0.02017372 0.64845241 0.02421308 0.00936553]
```

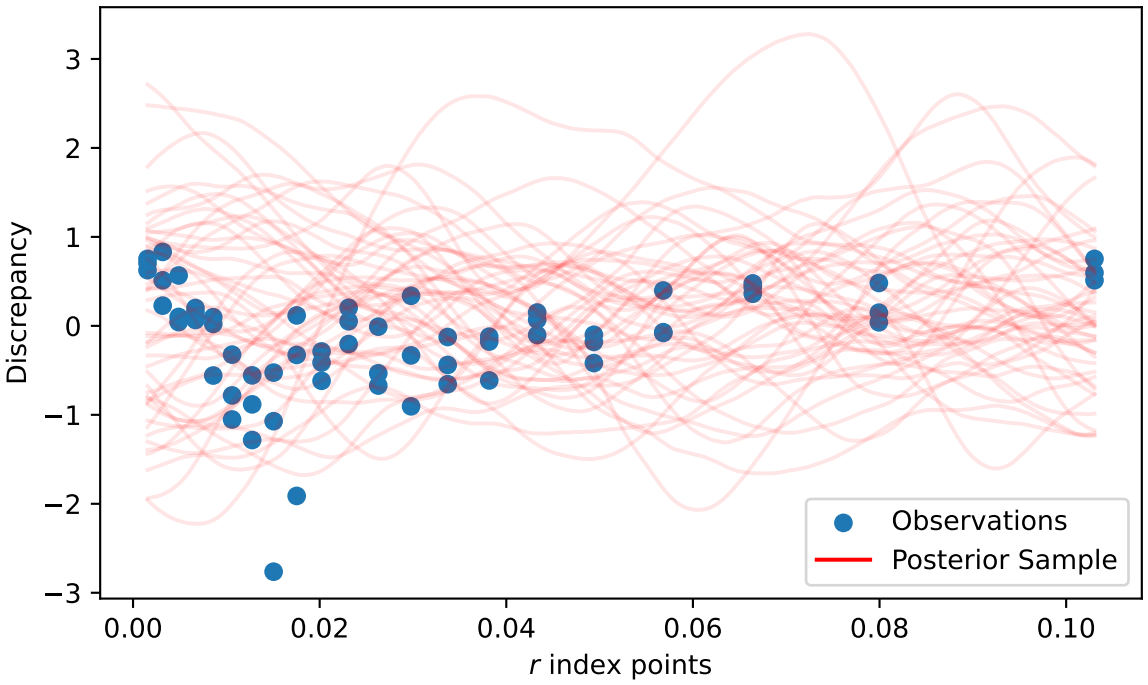


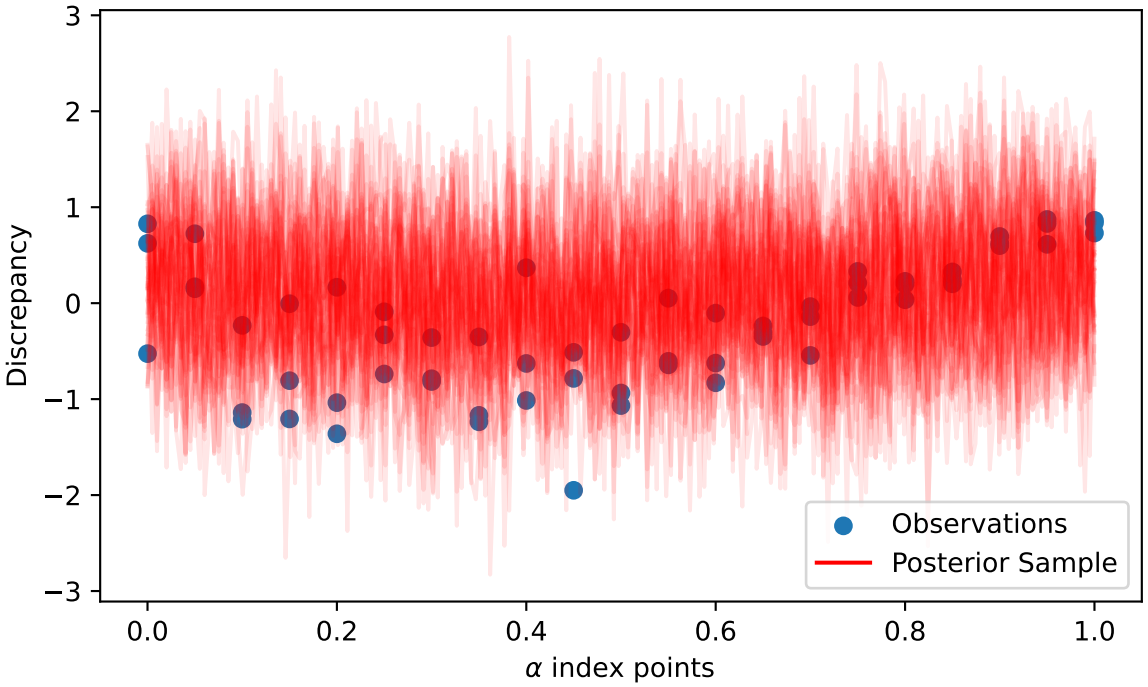

Discrepancy
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

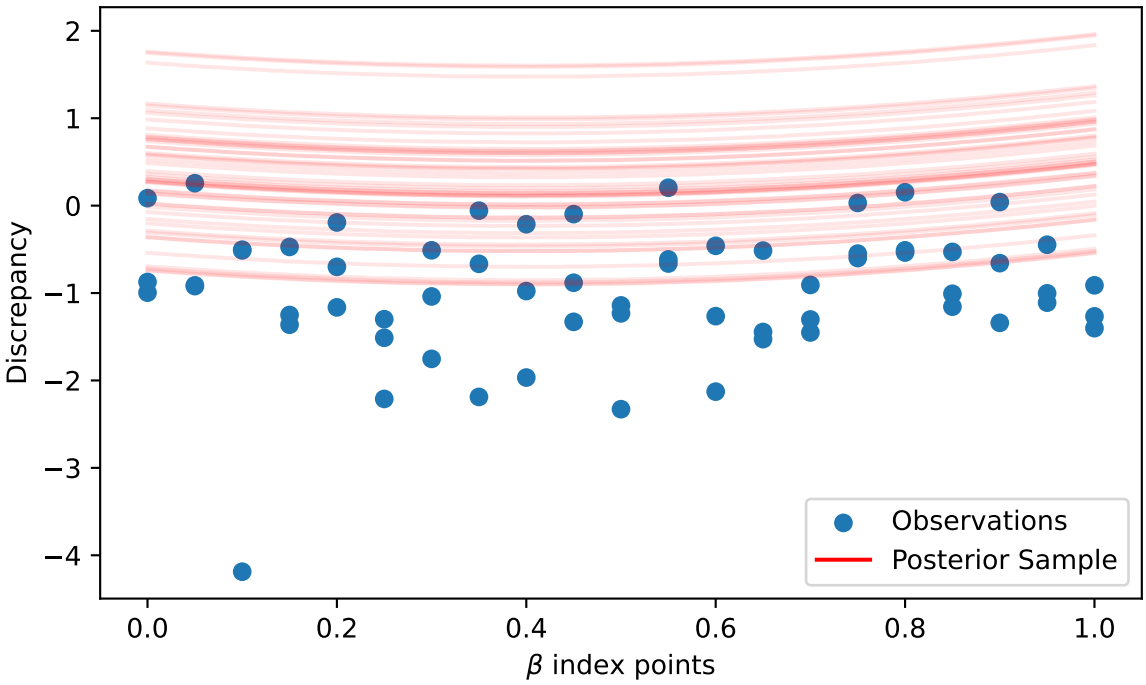

Discrepancy
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γL index points
Observations
Posterior Sample

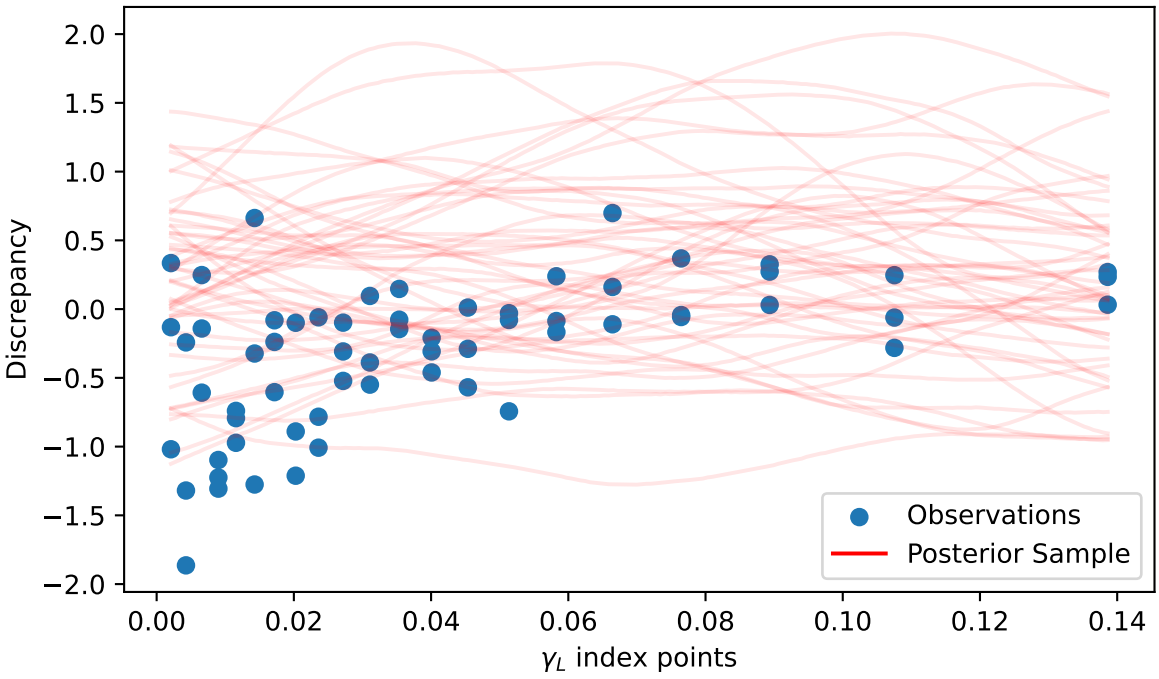

Discrepancy
4
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample

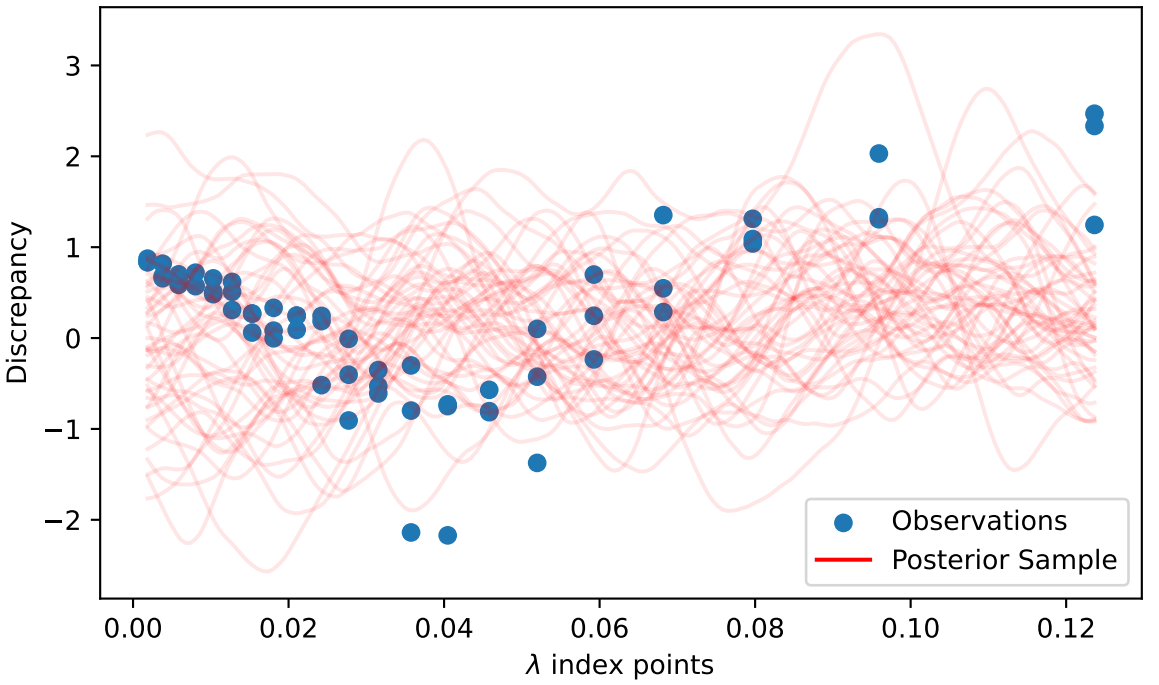

Discrepancy
2
1
0
−1
−2
0.0
0.1
0.2
0.3
0.4
f index points
Observations
Posterior Sample

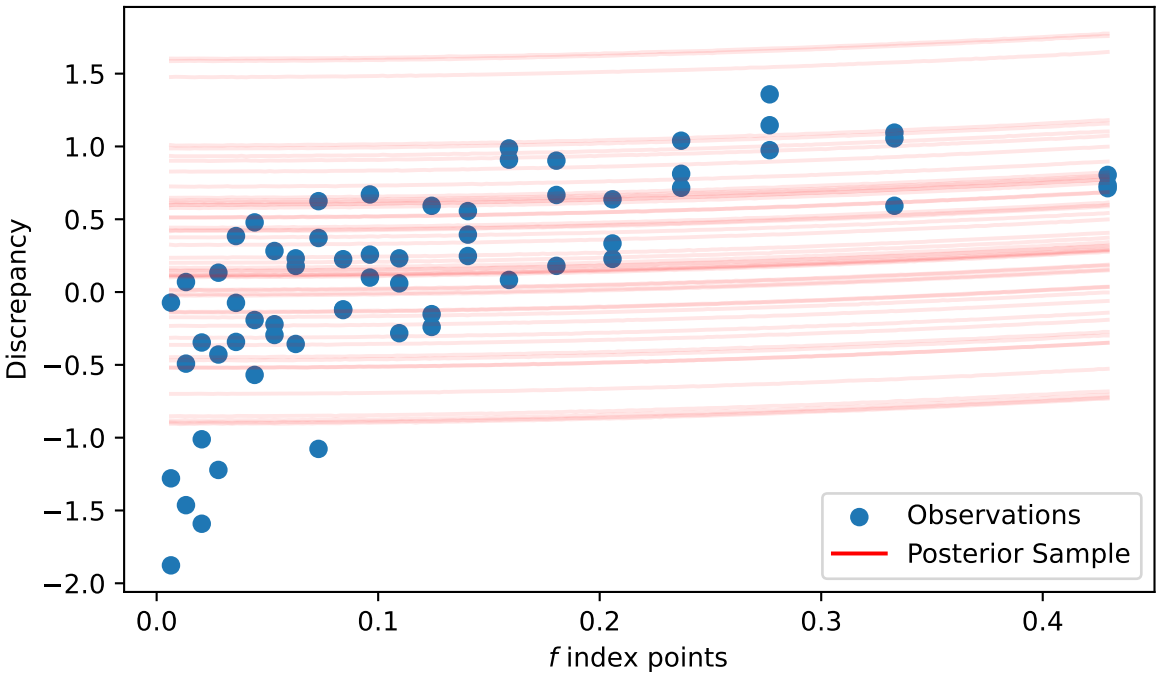

Discrepancy
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
r index points
Observations
Posterior Sample

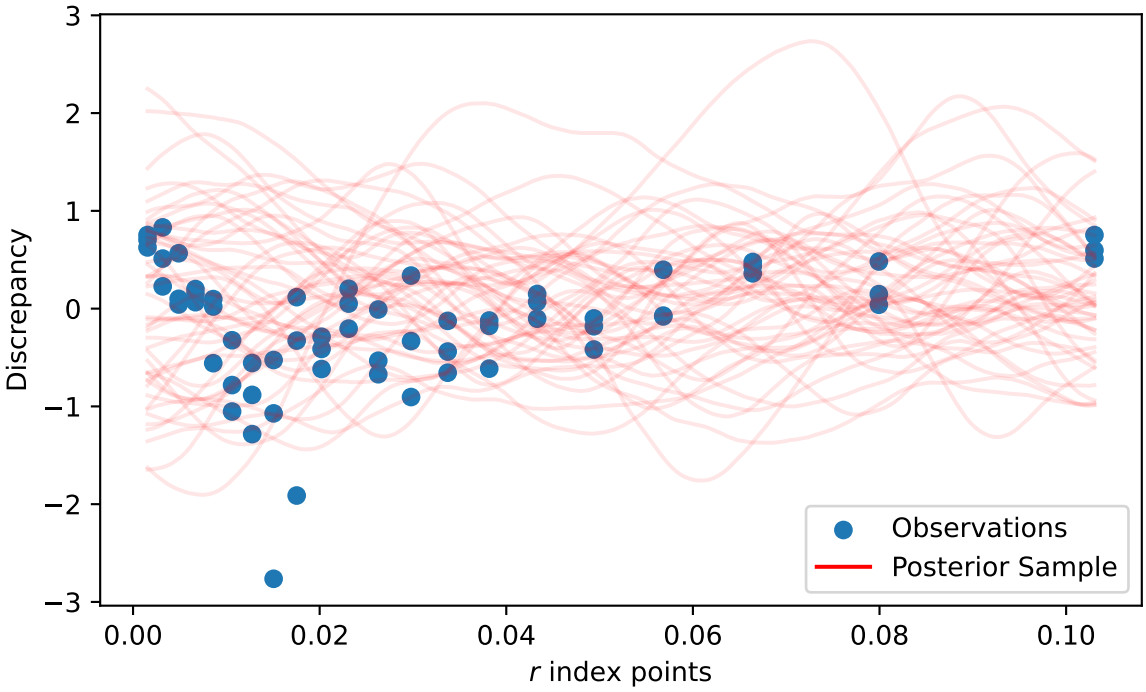

Discrepancy
3
2
1
0
−1
−2
−3
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample

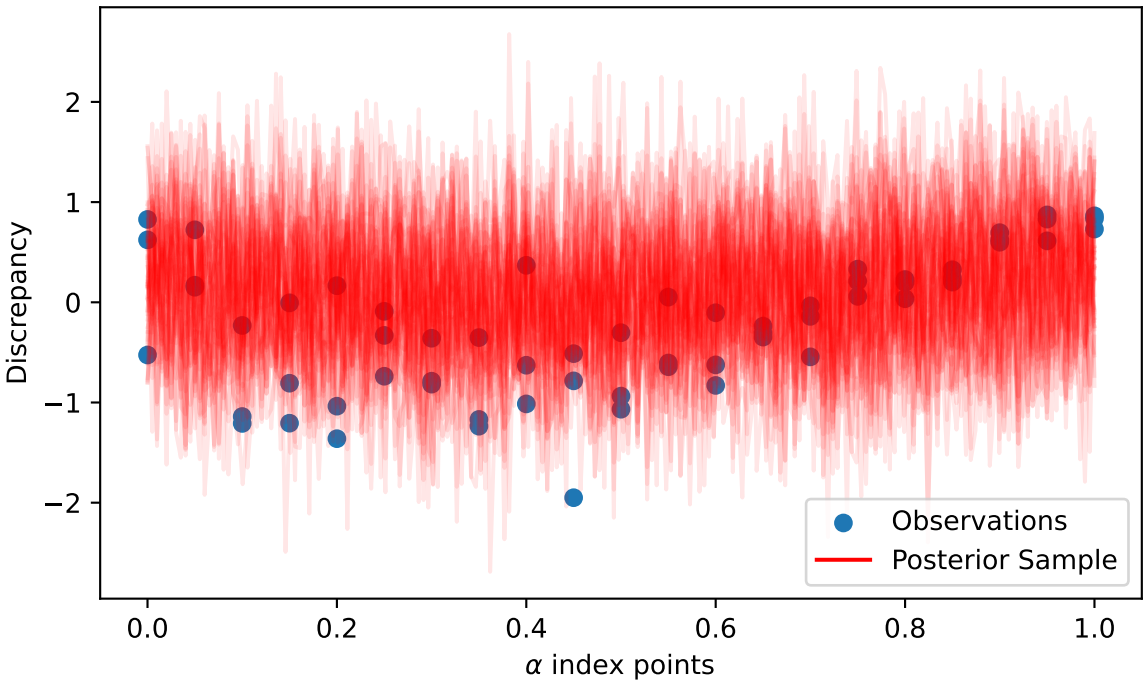

Discrepancy
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

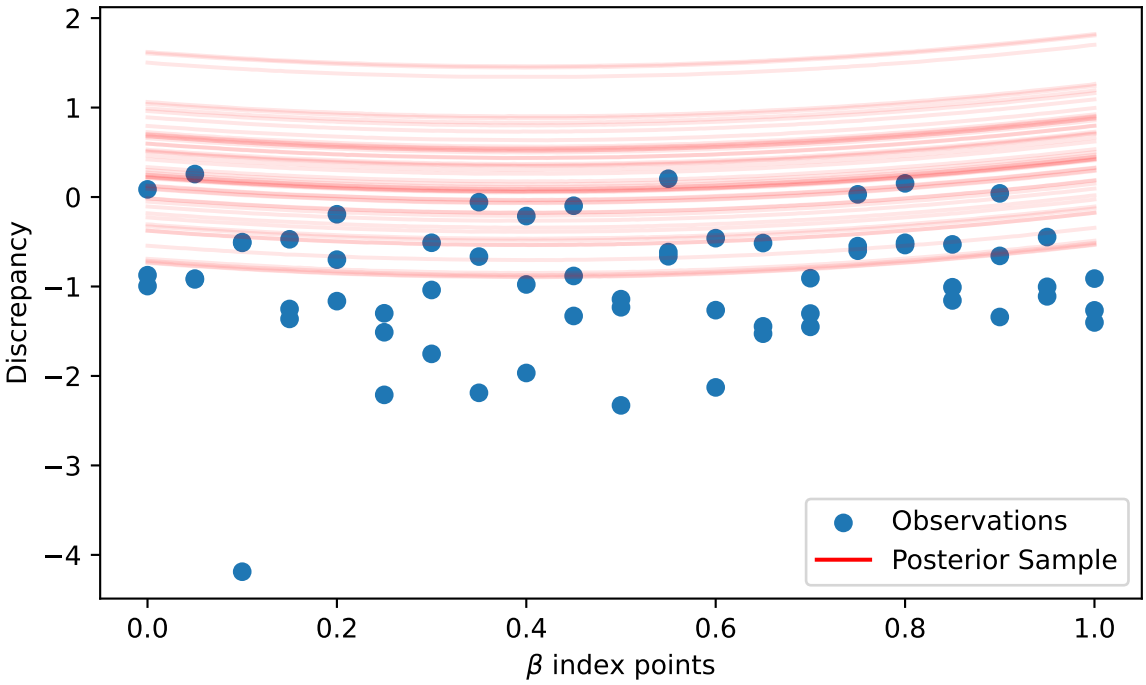

Discrepancy
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γ_L index points
Observations
Posterior Sample

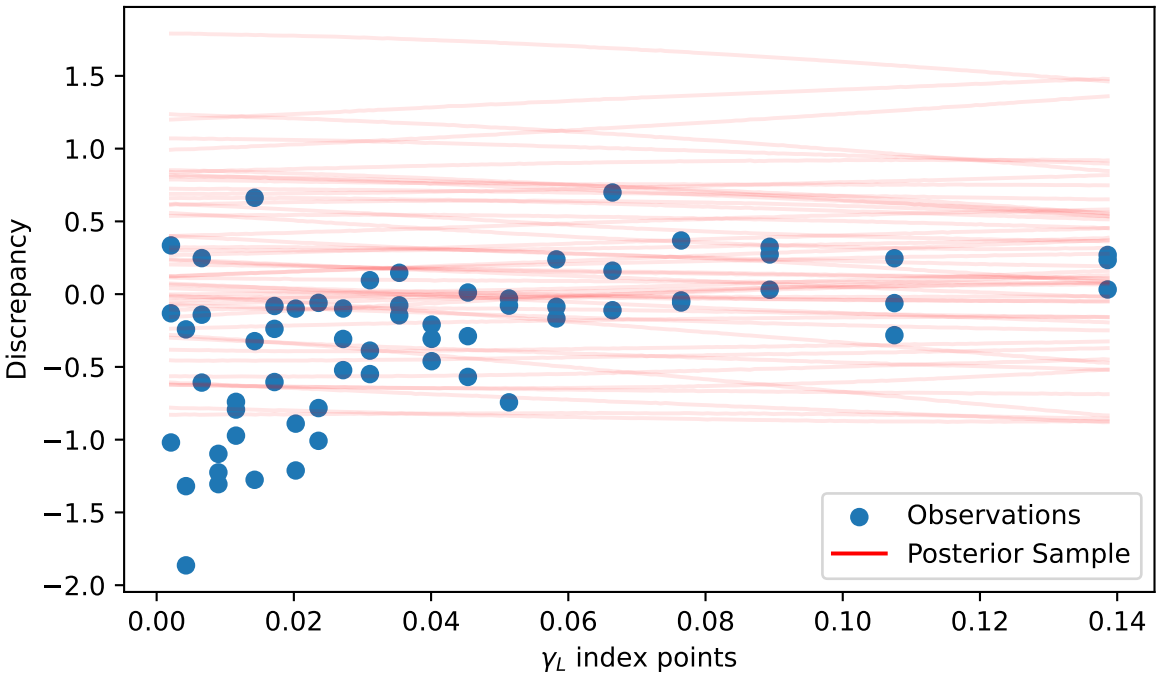

Discrepancy
3
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample

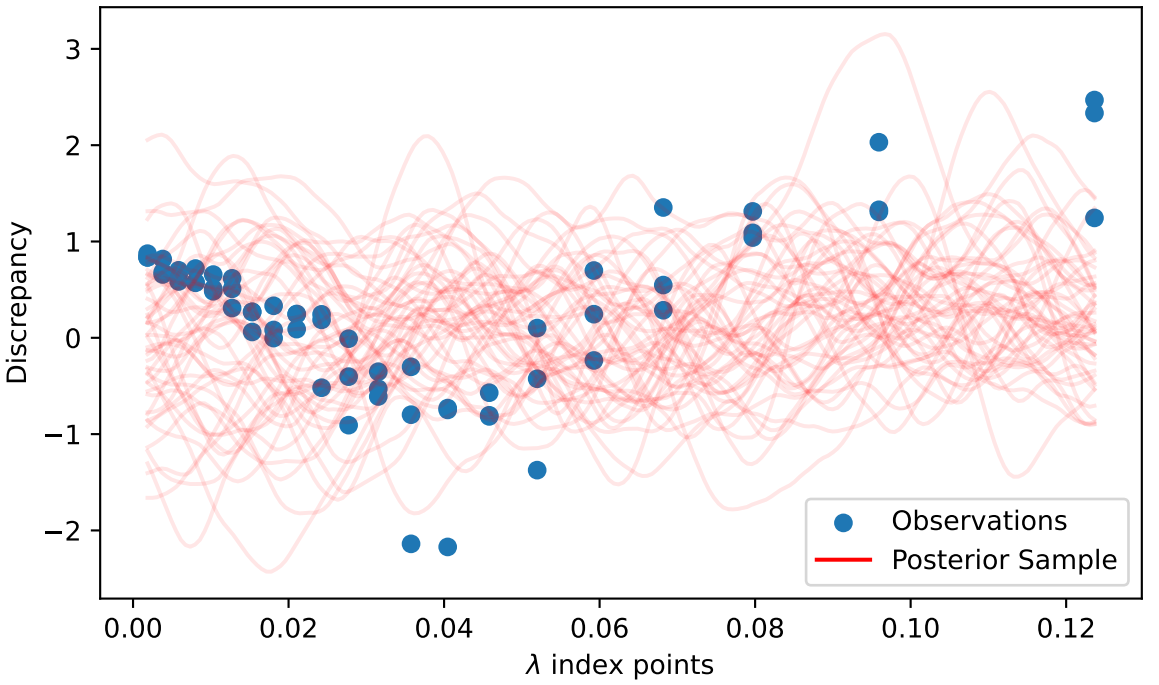

Discrepancy
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.1
0.2
0.3
0.4
f index points
Observations
Posterior Sample

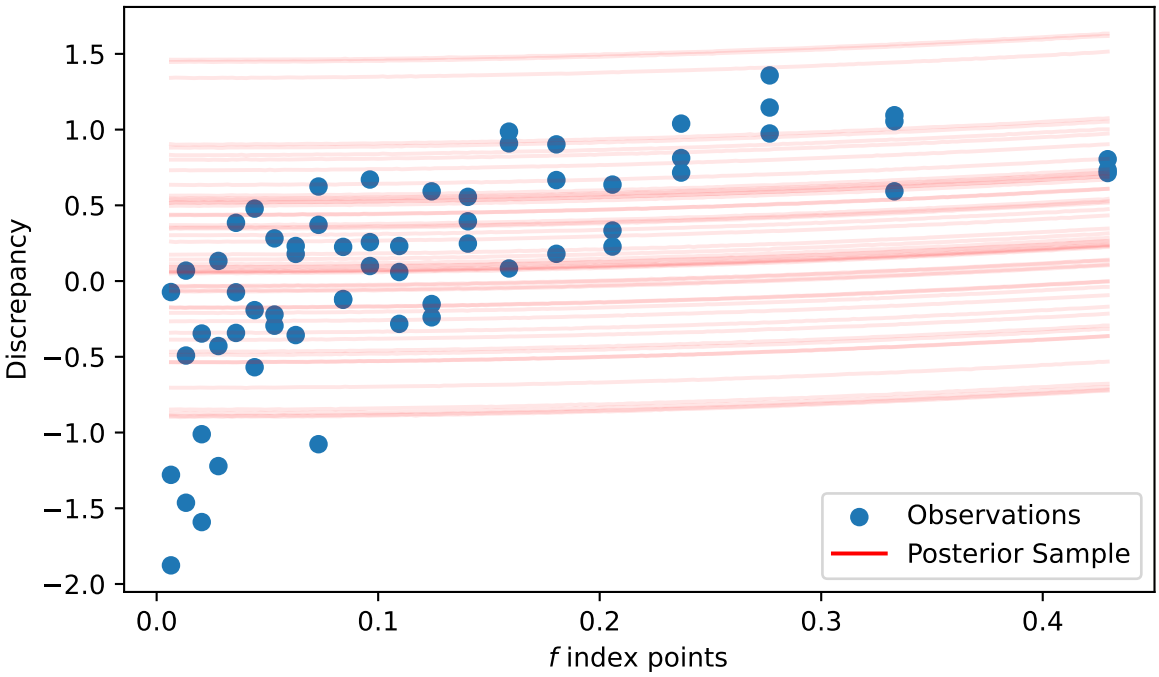

Discrepancy
3
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
r index points
Observations
Posterior Sample

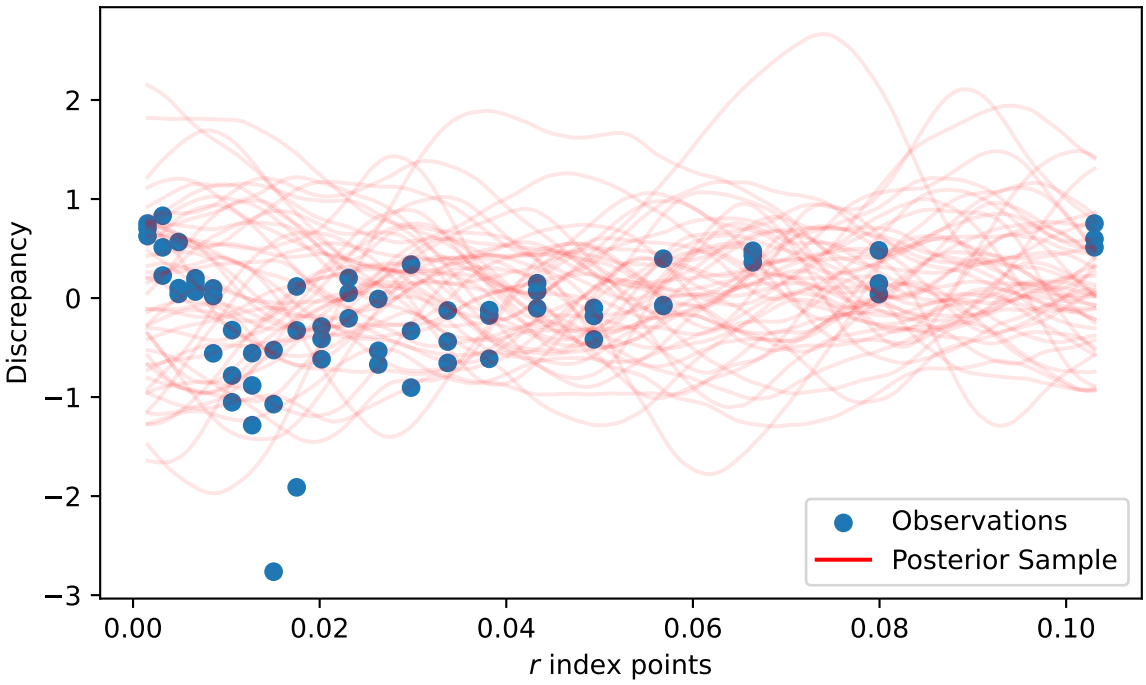

Discrepancy
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample

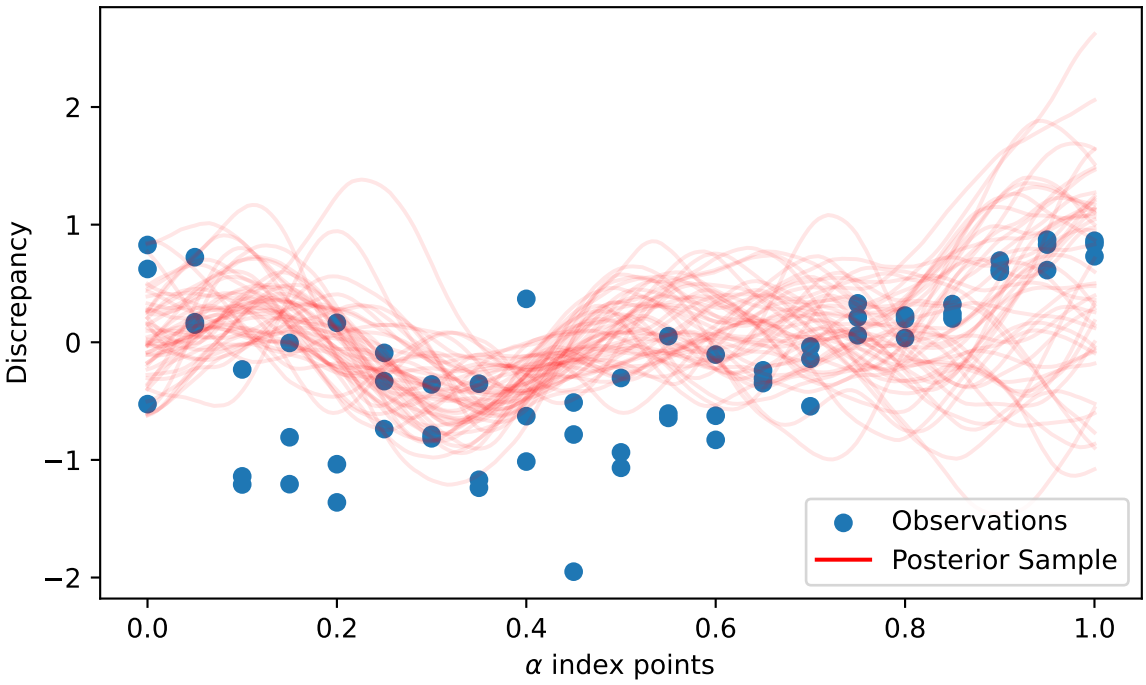

Discrepancy
2
1
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample

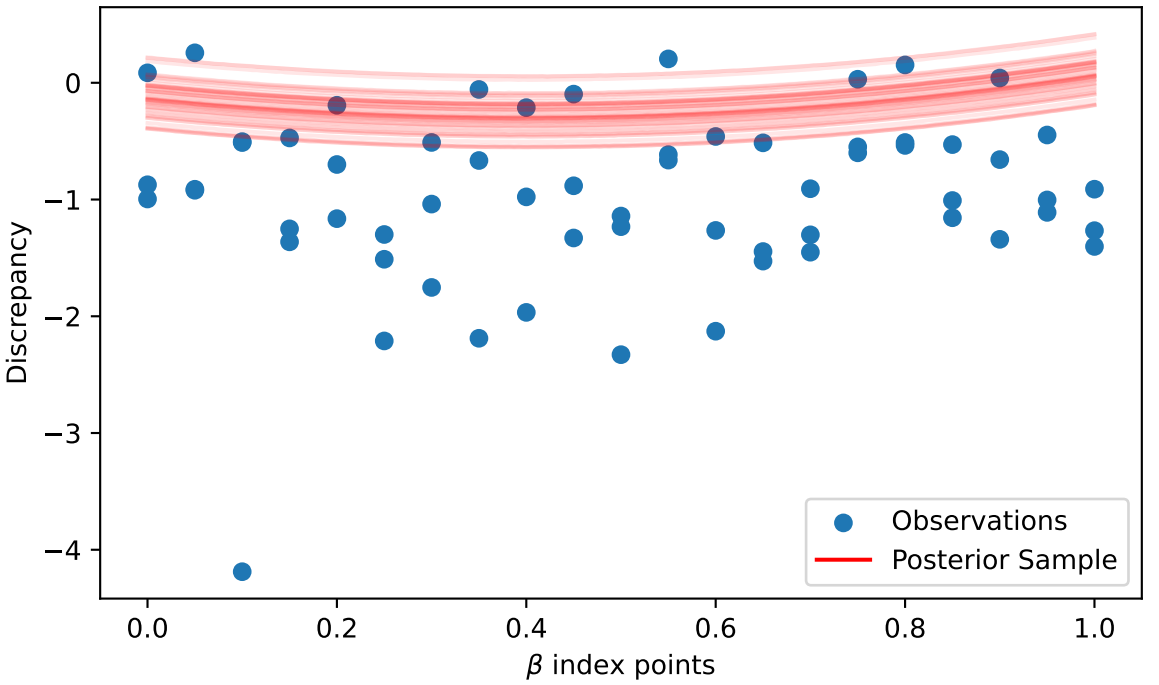

Discrepancy
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γL index points
Observations
Posterior Sample

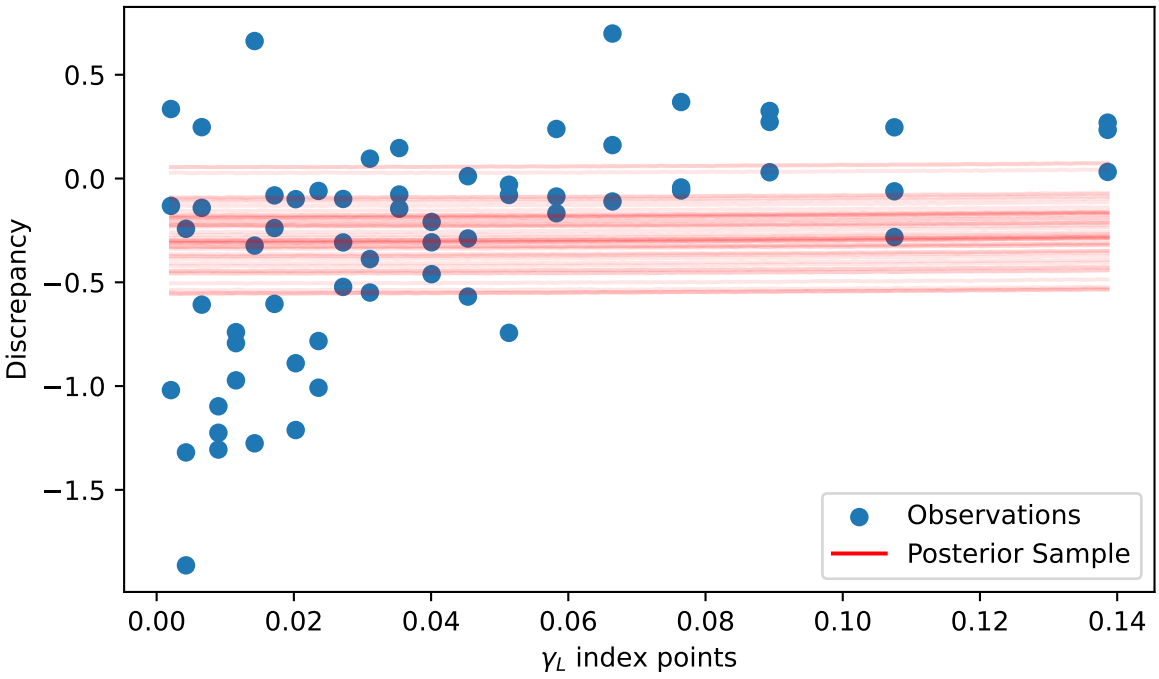

Discrepancy
3
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample

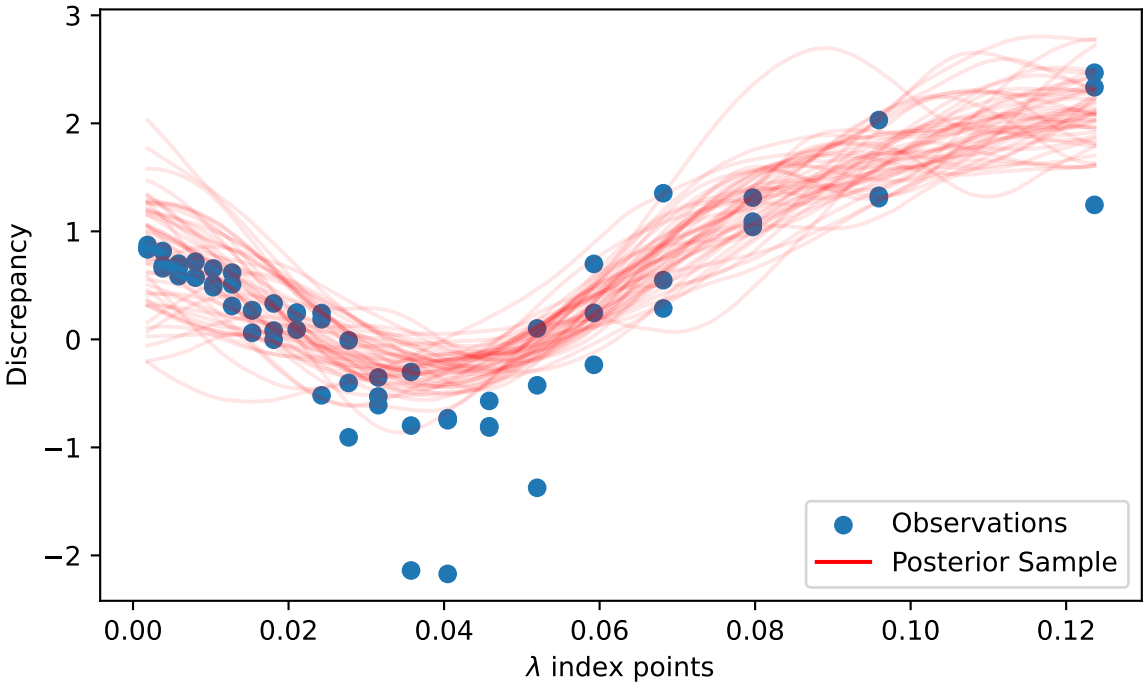

Discrepancy
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.1
0.2
0.3
0.4
f index points
Observations
Posterior Sample

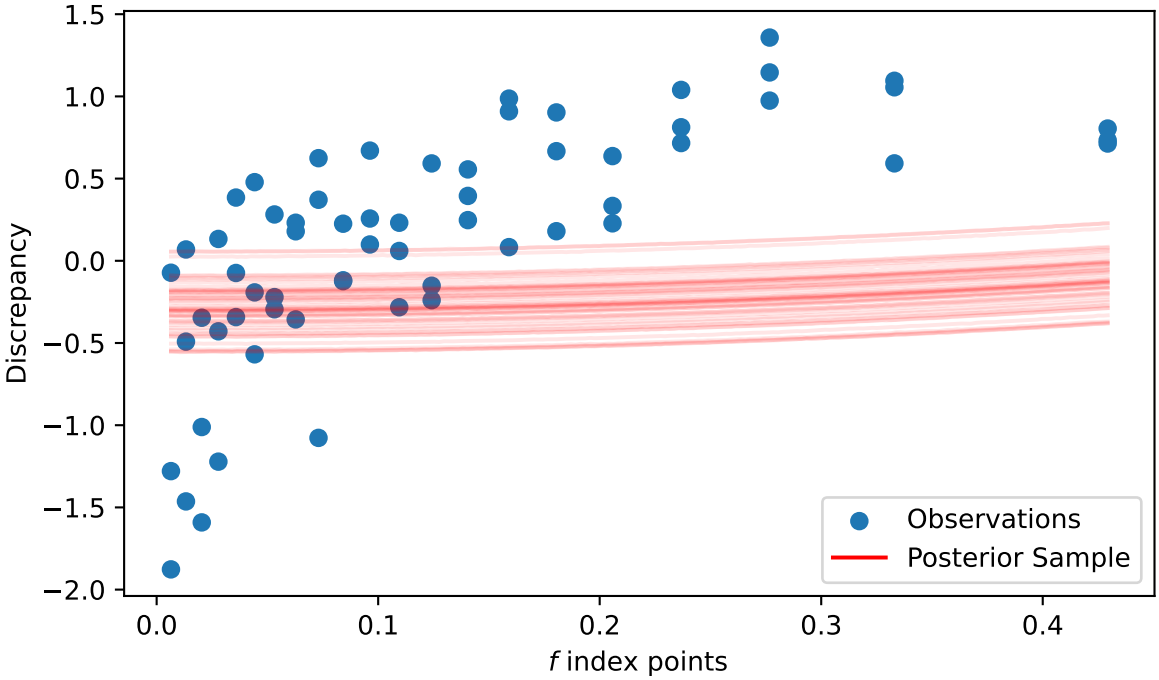

Discrepancy
2
1
0
−1
−2
−3
0.00
0.02
0.04
0.06
0.08
0.10
r index points
Observations
Posterior Sample

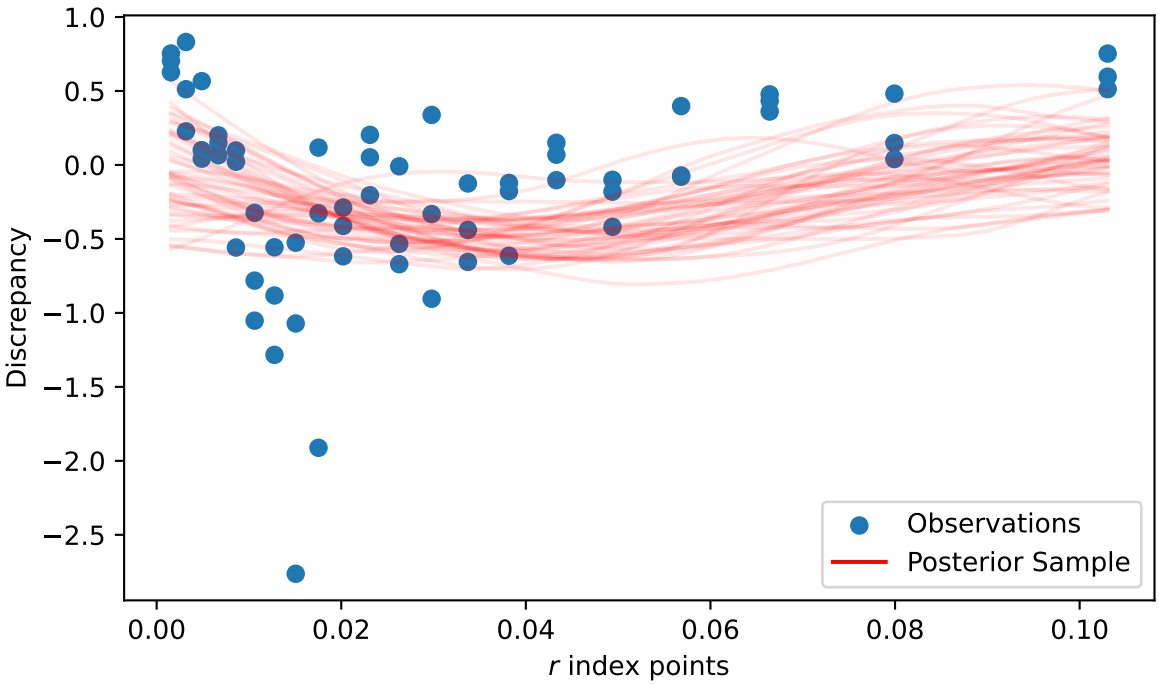

Discrepancy
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
α index points
Observations
Posterior Sample


Discrepancy
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
β index points
Observations
Posterior Sample


Discrepancy
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
γ_L index points
Observations
Posterior Sample


Discrepancy
3
2
1
0
−1
−2
0.00
0.02
0.04
0.06
0.08
0.10
0.12
λ index points
Observations
Posterior Sample


Discrepancy
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.1
0.2
0.3
0.4
f index points
Observations
Posterior Sample


1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Discrepancy
0.00
0.02
0.04
0.06
0.08
0.10
r index points
Observations
Posterior Sample