Some examples of Gaussian Processes

Imports

```
import sys
print(sys.executable)
```

```
/bin/python3
```

```
import numpy as np

import tensorflow as tf
import tensorflow_probability as tfp
tfb = tfp.bijectors
tfd = tfp.distributions
tfk = tfp.math.psd_kernels

import matplotlib.pyplot as plt
```

```
2024-04-16 12:18:21.599034: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not find c
2024-04-16 12:18:23.586504: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] U
2024-04-16 12:18:23.586572: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Un
2024-04-16 12:18:23.878122: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515]
2024-04-16 12:18:24.916884: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not find c
2024-04-16 12:18:24.920480: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tensorf
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with
2024-04-16 12:18:34.051700: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Wa
```

GP for Noiseless Cubic Target Function

```
my_seed = np.random.default_rng(seed=591) # For replicability

def cub_fn(x):
    return x[..., 0]*(x[..., 0] - 1)*(x[..., 0]+1)

index_vals = np.expand_dims( # this makes the size of the sample (3,1)
    my_seed.uniform(low=-1., high=1., size=5),
    axis=-1 # last axis
    )

obs_vals = cub_fn(index_vals)

my_GP = tfd.GaussianProcess(
      kernel=tfk.ExponentiatedQuadratic(
          amplitude=tf.Variable(1., dtype=np.float64, name="amplitude"),
          length_scale=tf.Variable(1., dtype=np.float64, name="length_scale")
      ),
      index_points=index_vals)

print(my_GP.trainable_variables[1])

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

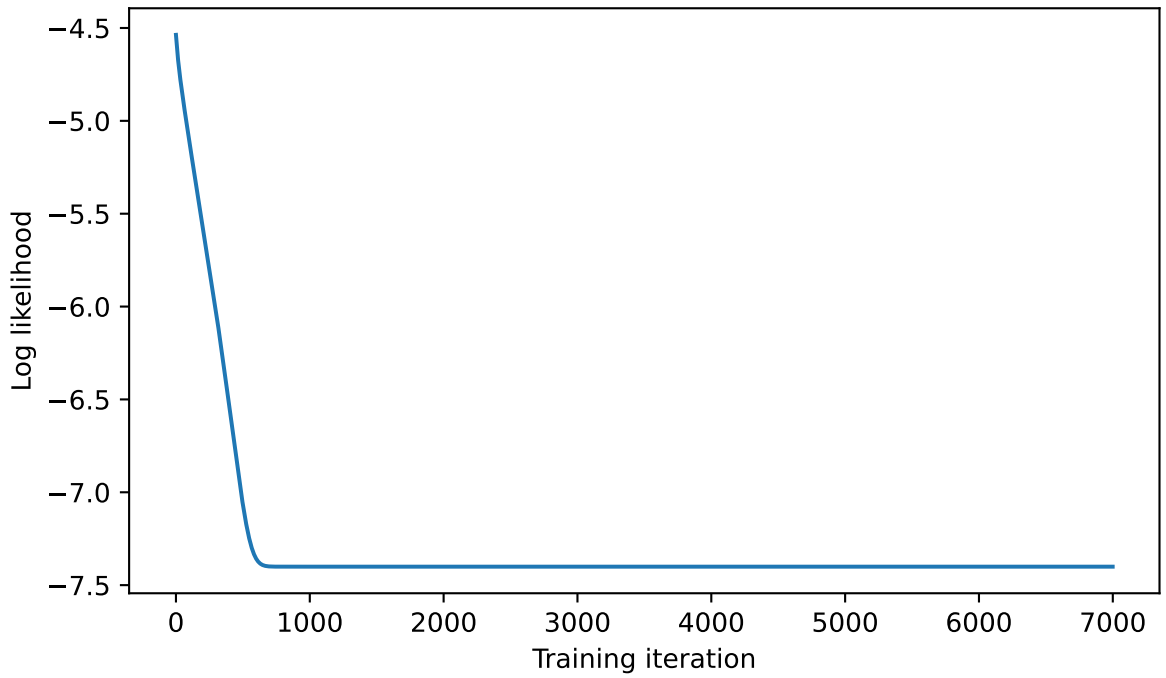
```
<tf.Variable 'length_scale:0' shape=() dtype=float64, numpy=1.0>
```

```
@tf.function()
def optimize():
  with tf.GradientTape() as tape:
    loss = -my_GP.log_prob(obs_vals)
  grads = tape.gradient(loss, my_GP.trainable_variables)
  Adam_optim.apply_gradients(zip(grads, my_GP.trainable_variables))
  return loss


num_iters = 7000

lls_ = np.zeros(num_iters, np.float64)
for i in range(num_iters):
```

```
  loss = optimize()
  lls_[i] = loss

amp_fin = my_GP.trainable_variables[0].numpy()
len_fin = my_GP.trainable_variables[1].numpy()

print('Trained parameters:')
print('amplitude: {}'.format(amp_fin))
print('length_scale: {}'.format(len_fin))
```

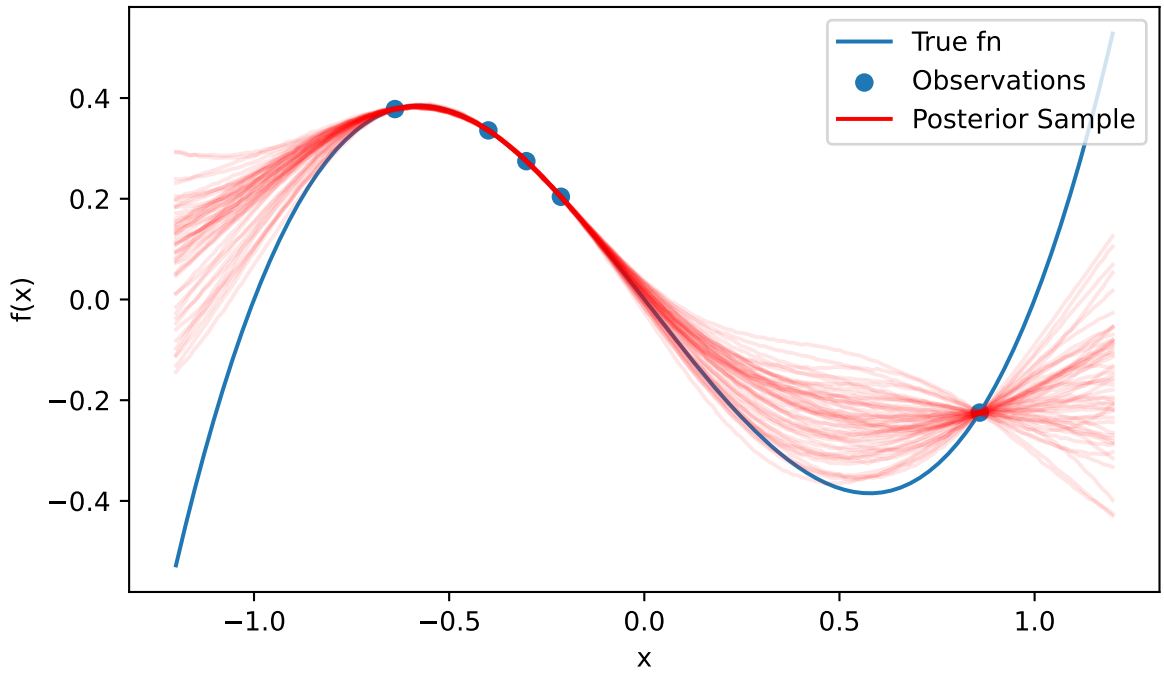
```
Trained parameters:
amplitude: 0.25353429007713696
length_scale: 0.5427608245129216
```

```
plt.figure(figsize=(7, 4))
plt.plot(lls_)
plt.xlabel("Training iteration")
plt.ylabel("Log likelihood")
plt.show()
```



```
# Having trained the model, we'd like to sample from the posterior conditioned
# on observations. We'd like the samples to be at points other than the training
# inputs.
predictive_index_points_ = np.linspace(-1.2, 1.2, 200, dtype=np.float64)
# Reshape to [200, 1] -- 1 is the dimensionality of the feature space.
predictive_index_points_ = predictive_index_points_[..., np.newaxis]

optimized_kernel = tfk.ExponentiatedQuadratic(amp_fin, len_fin)
gprm = tfd.GaussianProcessRegressionModel(
    kernel=optimized_kernel,
    index_points=predictive_index_points_,
    observation_index_points=index_vals,
    observations=obs_vals,
    predictive_noise_variance=0.)

# Create op to draw  50 independent samples, each of which is a *joint* draw
# from the posterior at the predictive_index_points_. Since we have 200 input
# locations as defined above, this posterior distribution over corresponding
# function values is a 200-dimensional multivariate Gaussian distribution!
num_samples = 50
samples = gprm.sample(num_samples)
```

```
plt.figure(figsize=(7, 4))
plt.plot(predictive_index_points_, cub_fn(predictive_index_points_),
         label='True fn')
plt.scatter(index_vals[:, 0], obs_vals,
            label='Observations')
for i in range(num_samples):
  plt.plot(predictive_index_points_, samples[i, :], c='r', alpha=.1,
           label='Posterior Sample' if i == 0 else None)
leg = plt.legend(loc='upper right')
for lh in leg.legend_handles:
    lh.set_alpha(1)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```



GP for Cubic Target Function with Noise

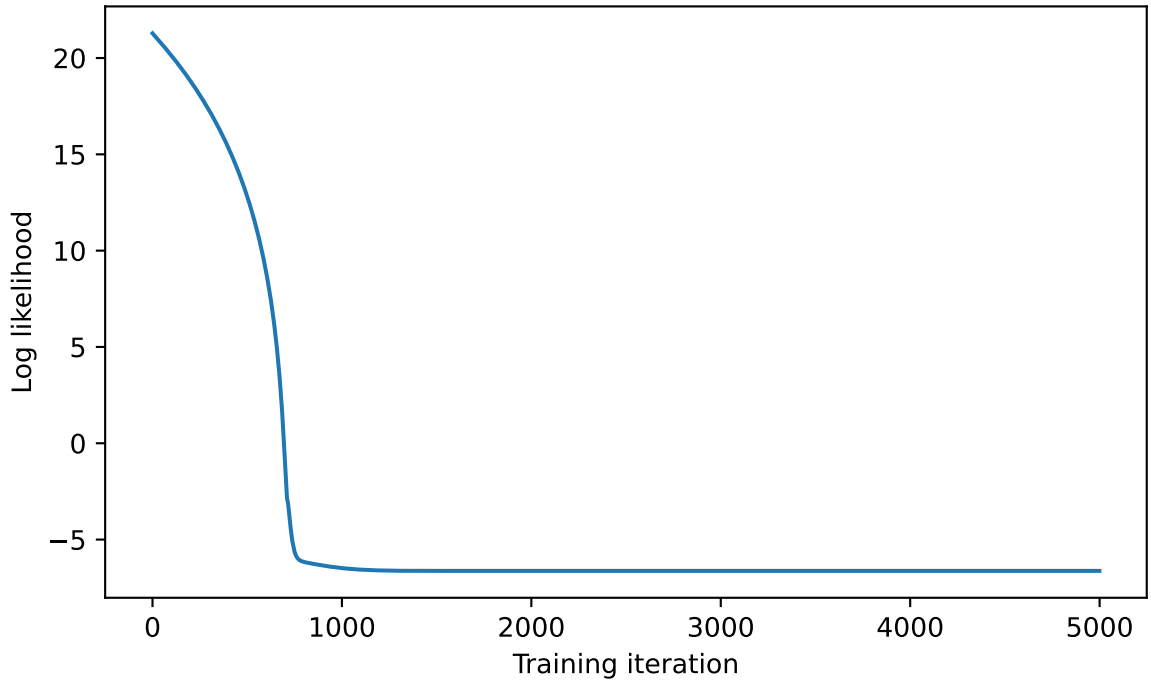
```
num_obs_err=20

my_seed_err=np.random.default_rng(seed=914)

def cub_fn_err(x):
    return x[..., 0]*(x[..., 0] - 1)*(x[..., 0]+1) + my_seed.normal(scale=.1, size=num_obs_e

index_vals_err = np.expand_dims( # this makes the size of the sample (3,1)
    my_seed_err.uniform(low=-1., high=1., size=num_obs_err),
    axis=-1 # last axis
    )

obs_vals_err = cub_fn_err(index_vals_err)

my_GP_err = tfd.GaussianProcess(
      kernel=tfk.ExponentiatedQuadratic(
          amplitude=tf.Variable(1., dtype=np.float64, name="amplitude_err"),
          length_scale=tf.Variable(1., dtype=np.float64, name="length_scale_err")
```

```
      ),
      observation_noise_variance=tf.Variable(1., dtype=np.float64, name="observation_noise_va
      index_points=index_vals_err)


print(my_GP_err.trainable_variables)
```

```
(<tf.Variable 'amplitude_err:0' shape=() dtype=float64, numpy=1.0>, <tf.Variable 'length_scal
```

```
Adam_optim = tf.optimizers.Adam() # somehow need this again? maybe when you do the optimize i

@tf.function()
def optimize_err():
  with tf.GradientTape() as tape:
    loss = -my_GP_err.log_prob(obs_vals_err)
  grads = tape.gradient(loss, my_GP_err.trainable_variables)
  Adam_optim.apply_gradients(zip(grads, my_GP_err.trainable_variables))
  return loss


num_iters_err = 5000

lls_err = np.zeros(num_iters_err, np.float64)
for i in range(num_iters_err):
  loss_err = optimize_err()
  lls_err[i] = loss_err

amp_fin_err = my_GP_err.trainable_variables[0].numpy()
len_fin_err = my_GP_err.trainable_variables[1].numpy()
obs_var_fin_err = my_GP_err.trainable_variables[2].numpy()

print('Trained parameters:')
print('amplitude_err: {}'.format(amp_fin))
print('length_scale_err: {}'.format(len_fin))
print('observation_noise_variance_err: {}'.format(obs_var_fin_err))

plt.figure(figsize=(7, 4))
plt.plot(lls_err)
plt.xlabel("Training iteration")
plt.ylabel("Log likelihood")
plt.show()
```

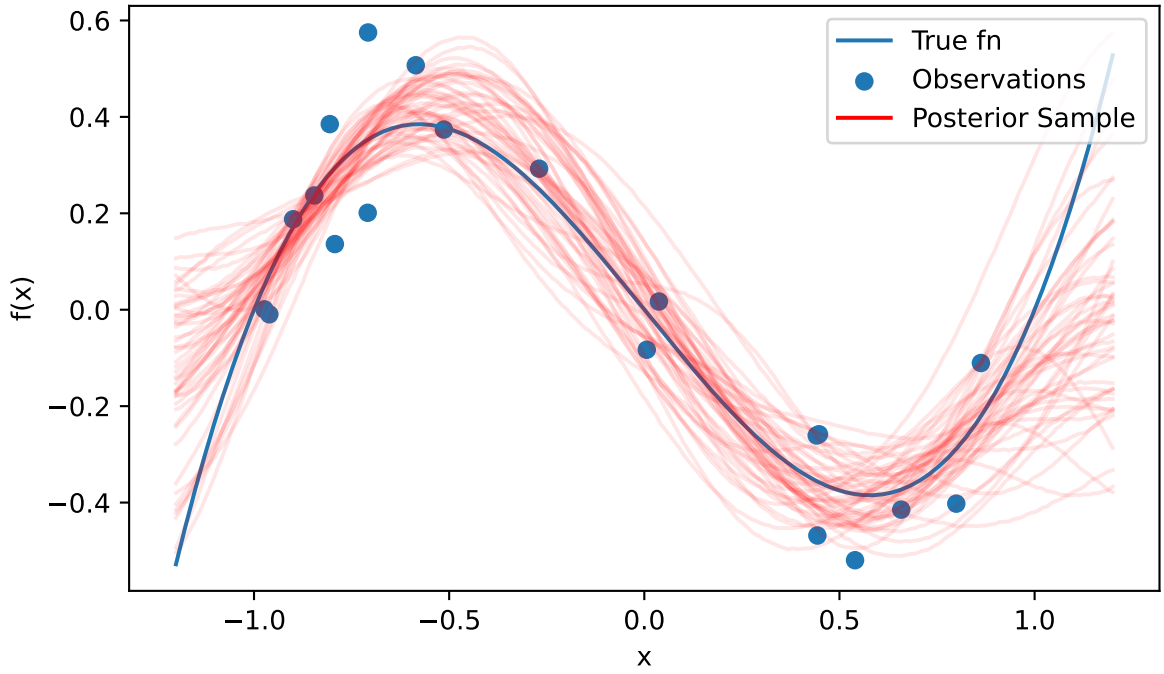
```
Trained parameters:
amplitude_err: 0.25353429007713696
length_scale_err: 0.5427608245129216
observation_noise_variance_err: 0.014781696799272675
```




```
optimized_kernel_err = tfk.ExponentiatedQuadratic(amp_fin_err, len_fin_err)
gprm_err = tfd.GaussianProcessRegressionModel(
    kernel=optimized_kernel_err,
    index_points=predictive_index_points_,
    observation_index_points=index_vals_err,
    observations=obs_vals_err,
    observation_noise_variance=obs_var_fin_err,
    predictive_noise_variance=0.)

num_samples_err = 50
samples_err = gprm_err.sample(num_samples_err)
```

```
plt.figure(figsize=(7, 4))
plt.plot(predictive_index_points_, cub_fn(predictive_index_points_),
         label='True fn')
plt.scatter(index_vals_err[:, 0], obs_vals_err,
```

```
            label='Observations')
for i in range(num_samples_err):
  plt.plot(predictive_index_points_, samples_err[i, :], c='r', alpha=.1,
           label='Posterior Sample' if i == 0 else None)
leg = plt.legend(loc='upper right')
for lh in leg.legend_handles:
    lh.set_alpha(1)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```



GP with Quadratic Mean Function for Cubic Target Function

```
num_obs_quad=4

my_seed_quad=np.random.default_rng(seed=687)


index_vals_quad = np.expand_dims( # this makes the size of the sample (3,1)
```

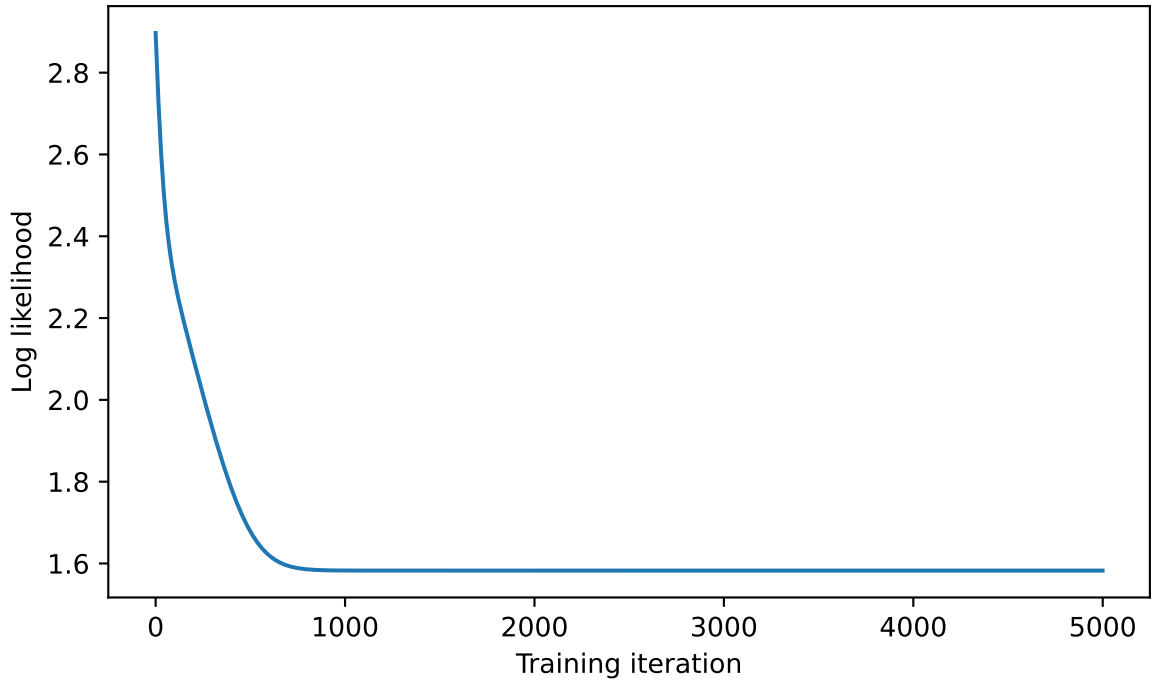
```
    my_seed_quad.uniform(low=-1., high=1., size=num_obs_quad),
    axis=-1 # last axis
    )

obs_vals_quad = cub_fn(index_vals_quad)

def custom_mean_fn(x):
    # You can define your own mean function here
    # For example, a linear mean function: return 2.0 * x
    return - x[..., 0]**2

my_GP_quad = tfd.GaussianProcess(
      kernel=tfk.ExponentiatedQuadratic(
          amplitude=tf.Variable(1., dtype=np.float64, name="amplitude_quad"),
          length_scale=tf.Variable(1., dtype=np.float64, name="length_scale_quad")
      ),
      index_points=index_vals_quad,
      mean_fn=custom_mean_fn)

print(my_GP_quad.trainable_variables)
```

```
(<tf.Variable 'amplitude_quad:0' shape=() dtype=float64, numpy=1.0>, <tf.Variable 'length_sca
```

```
Adam_optim = tf.optimizers.Adam() # somehow need this again? maybe when you do the optimize i

@tf.function()
def optimize_quad():
  with tf.GradientTape() as tape:
    loss = -my_GP_quad.log_prob(obs_vals_quad)
  grads = tape.gradient(loss, my_GP_quad.trainable_variables)
  Adam_optim.apply_gradients(zip(grads, my_GP_quad.trainable_variables))
  return loss


num_iters_quad = 5000

lls_quad = np.zeros(num_iters_quad, np.float64)
for i in range(num_iters_quad):
  loss_quad = optimize_quad()
  lls_quad[i] = loss_quad
```

```
amp_fin_quad = my_GP_quad.trainable_variables[0].numpy()
len_fin_quad = my_GP_quad.trainable_variables[1].numpy()

print('Trained parameters:')
print('amplitude_quad: {}'.format(amp_fin))
print('length_scale_quad: {}'.format(len_fin))

plt.figure(figsize=(7, 4))
plt.plot(lls_quad)
plt.xlabel("Training iteration")
plt.ylabel("Log likelihood")
plt.show()
```

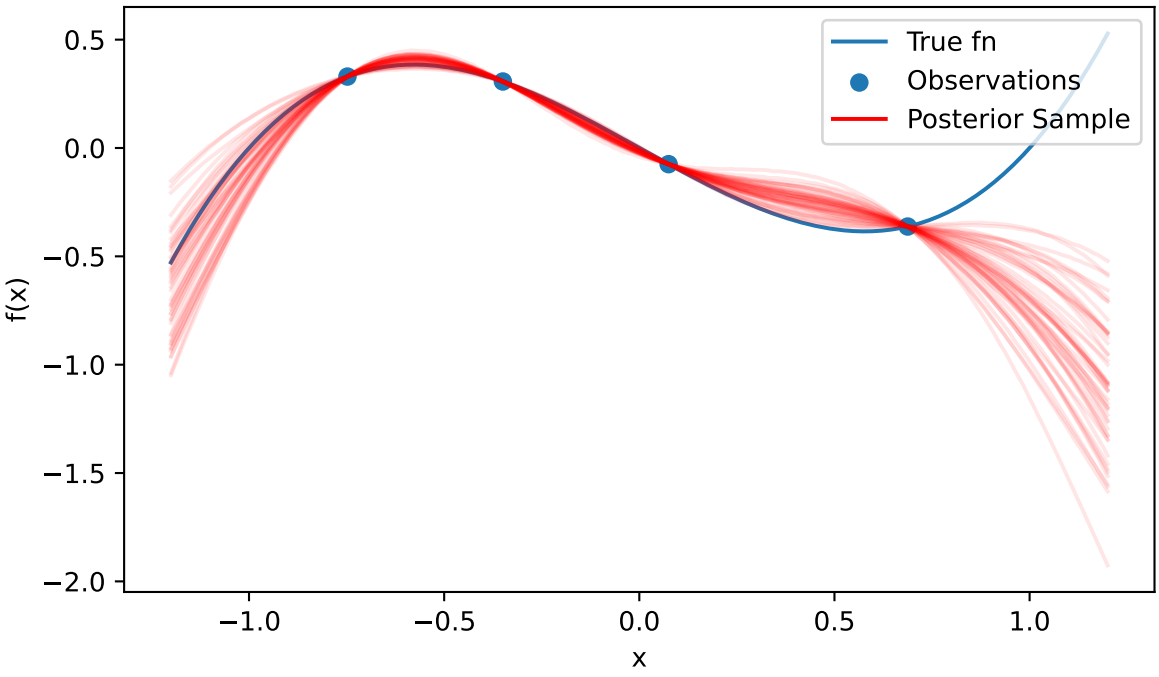
```
Trained parameters:
amplitude_quad: 0.25353429007713696
length_scale_quad: 0.5427608245129216
```




```
optimized_kernel_quad = tfk.ExponentiatedQuadratic(amp_fin_quad, len_fin_quad)
gprm_quad = tfd.GaussianProcessRegressionModel(
    kernel=optimized_kernel_quad,
```

```
    index_points=predictive_index_points_,
    observation_index_points=index_vals_quad,
    observations=obs_vals_quad,
    predictive_noise_variance=0.,
    mean_fn=custom_mean_fn)

num_samples_quad = 50
samples_quad = gprm_quad.sample(num_samples_quad)
```

```
plt.figure(figsize=(7, 4))
plt.plot(predictive_index_points_, cub_fn(predictive_index_points_),
         label='True fn')
plt.scatter(index_vals_quad[:, 0], obs_vals_quad,
            label='Observations')
for i in range(num_samples_quad):
  plt.plot(predictive_index_points_, samples_quad[i, :], c='r', alpha=.1,
           label='Posterior Sample' if i == 0 else None)
leg = plt.legend(loc='upper right')
for lh in leg.legend_handles:
    lh.set_alpha(1)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```

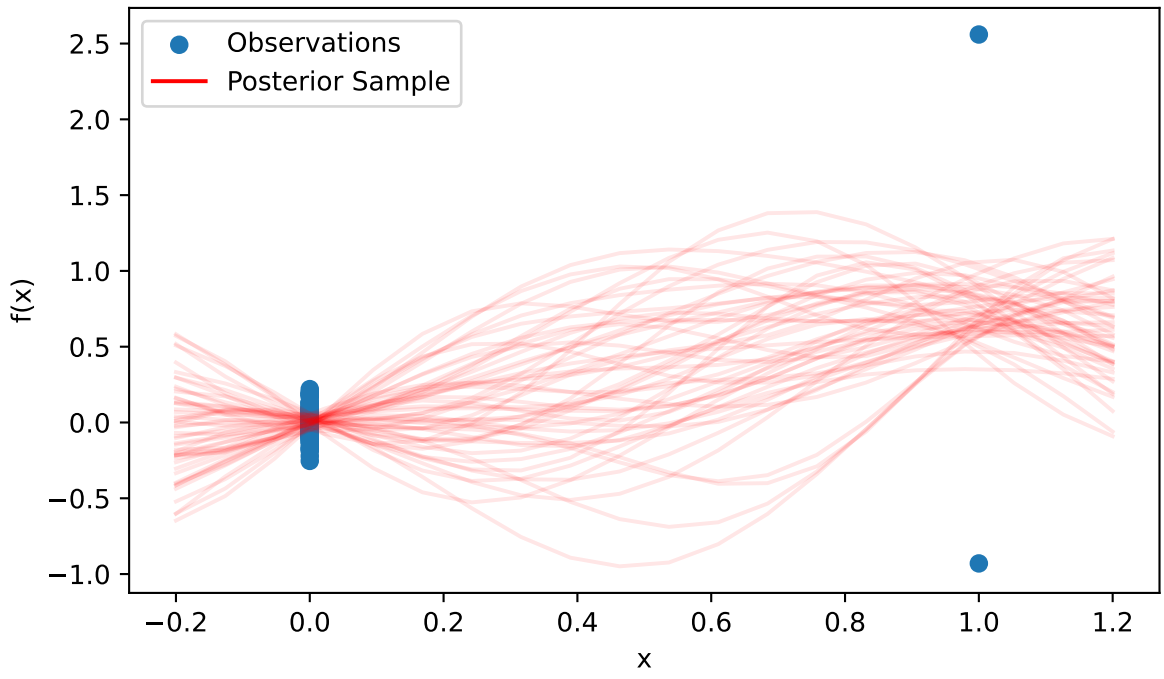


GP for 2 observations with Differing Observation Variance

```
def varying_variance(num_low, num_high, var_low, var_high):
  seed_low=np.random.default_rng(seed=973)
  seed_high=np.random.default_rng(seed=8973)
  seed_other=np.random.default_rng(seed=873)

  samples_low_ = seed_low.normal(scale = np.sqrt(var_low), size = num_low) #+ np.ones(num_lo
  samples_high_ = seed_high.normal(scale = np.sqrt(var_high), size = num_high) + np.ones(num_

  x_indices_ = np.expand_dims( # this makes the size of the sample (3,1)
    np.concatenate((np.zeros(num_low), np.ones(num_high))),
    axis=-1 # last axis
    )
  y_vals_ = np.concatenate((samples_low_, samples_high_))

  GP_ = tfd.GaussianProcess(
      kernel=tfk.ExponentiatedQuadratic(
          amplitude=tf.Variable(1., dtype=np.float64, name="amplitude"),
```

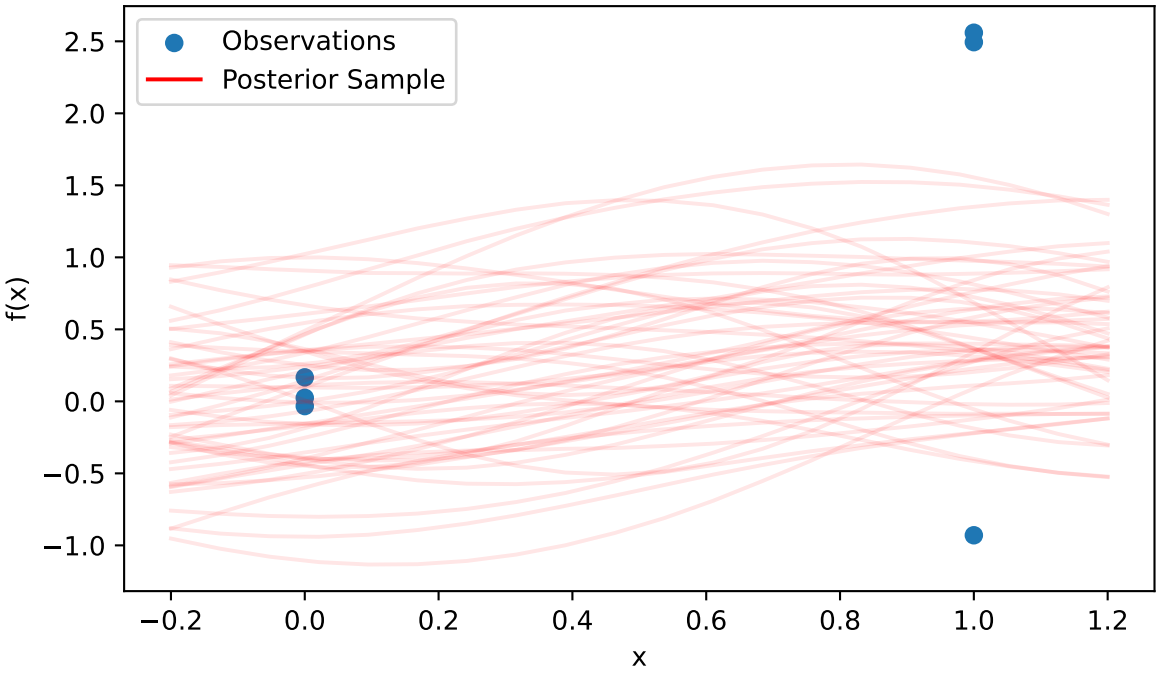
```
        length_scale=tf.Variable(1., dtype=np.float64, name="length_scale")
    ),
    observation_noise_variance=tf.Variable(1., dtype=np.float64, name="observation_noise_va
    index_points=x_indices_
    )

optimizer_ = tf.optimizers.Adam() # somehow need this again? maybe when you do the optimize

@tf.function()
def optimize_():
  with tf.GradientTape() as tape:
    loss = -GP_.log_prob(y_vals_)
  grads = tape.gradient(loss, GP_.trainable_variables)
  optimizer_.apply_gradients(zip(grads, GP_.trainable_variables))
  return loss


num_iters_ = 5000

lls_ = np.zeros(num_iters_, np.float64)
for i in range(num_iters_):
  loss_ = optimize_()
  lls_[i] = loss_

amp_fin_ = GP_.trainable_variables[0].numpy()
len_fin_ = GP_.trainable_variables[1].numpy()
noise_fin_ = GP_.trainable_variables[2].numpy()

predictive_index_points_ = np.linspace(-.2, 1.2, 20, dtype=np.float64)
# Reshape to [200, 1] -- 1 is the dimensionality of the feature space.
predictive_index_points_ = predictive_index_points_[..., np.newaxis]

optimized_kernel_ = tfk.ExponentiatedQuadratic(amp_fin_, len_fin_)
gprm_ = tfd.GaussianProcessRegressionModel(
    kernel=optimized_kernel_,
    index_points=predictive_index_points_,
    observation_index_points=x_indices_,
    observations=y_vals_,
    observation_noise_variance=noise_fin_,
    predictive_noise_variance=0.)

num_fn_samples_ = 50
```

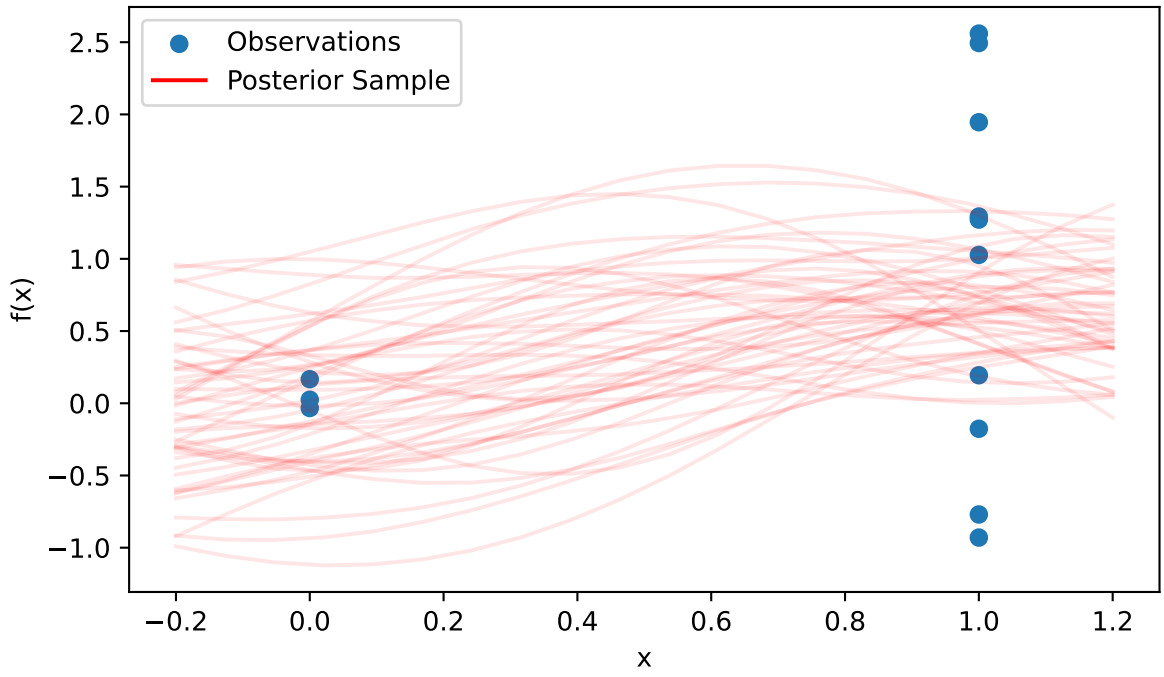
```
  tf.random.set_seed(25626)
  samples_ = gprm_.sample(num_fn_samples_)
  plt.figure(figsize=(7, 4))
  plt.scatter(x_indices_[:, 0], y_vals_,
              label='Observations')
  for i in range(num_fn_samples_):
    plt.plot(predictive_index_points_, samples_[i, :], c='r', alpha=.1,
             label='Posterior Sample' if i == 0 else None)
  leg = plt.legend(loc='upper left')
  for lh in leg.legend_handles:
      lh.set_alpha(1)
  plt.xlabel("x")
  plt.ylabel("f(x)")
  plt.show()

  return amp_fin_, len_fin_, noise_fin_

varying_variance(100, 2, 0.01, 1)
```

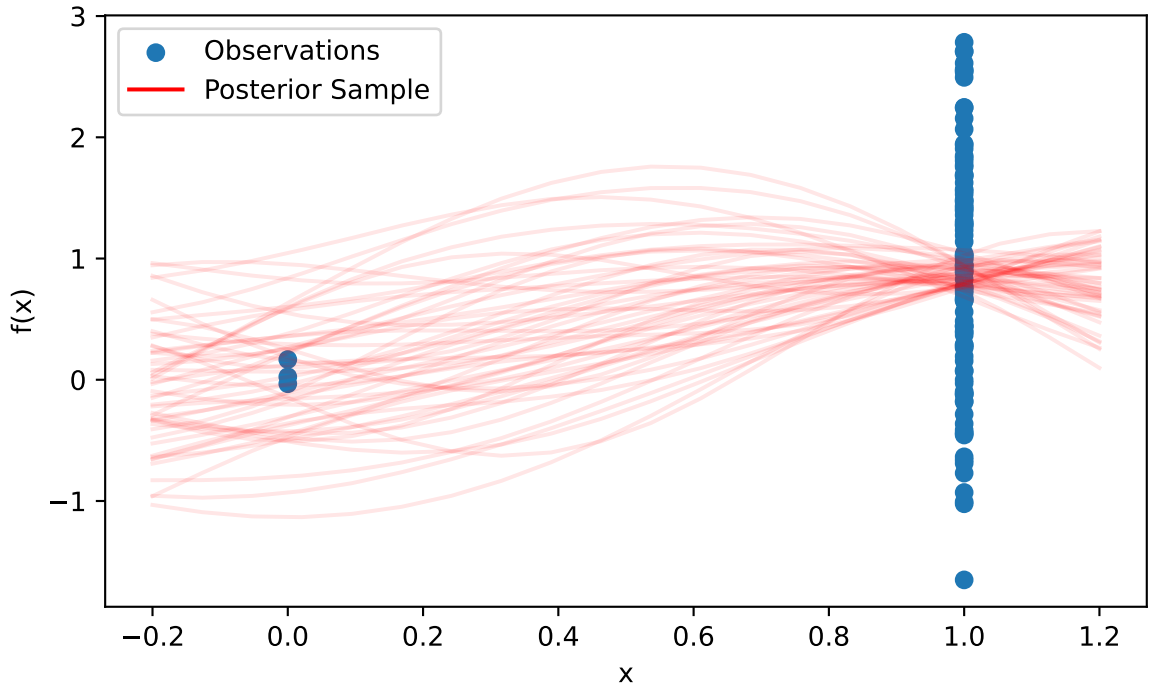


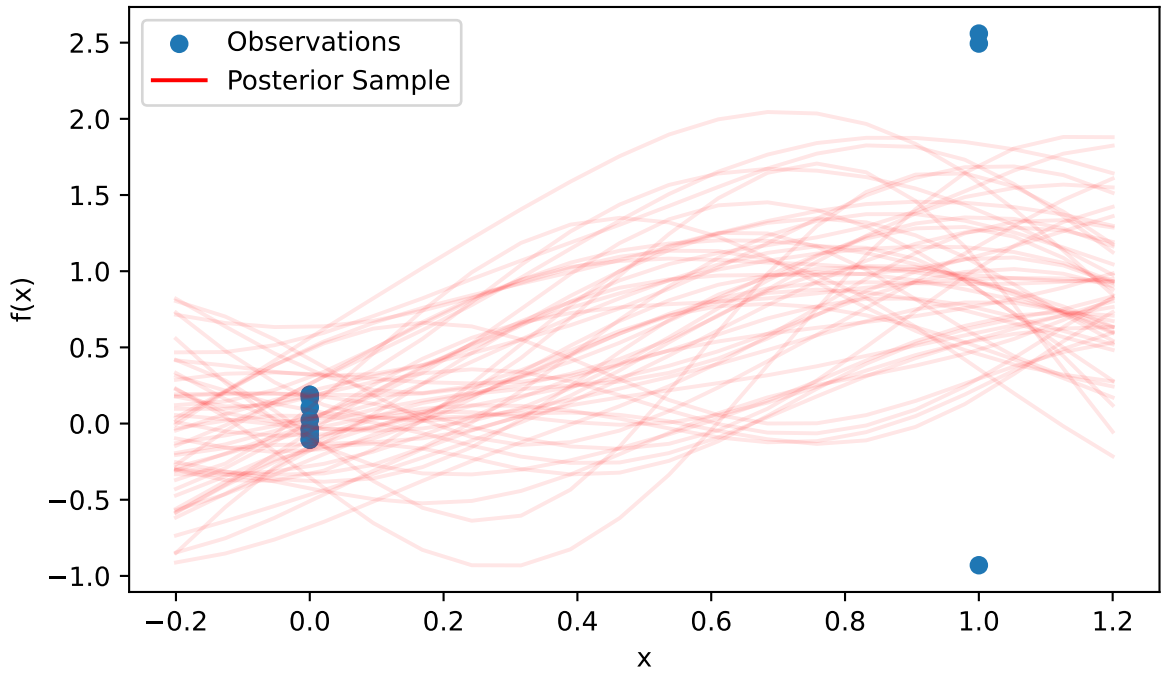
Experimenting with Differing Numbers of Observations

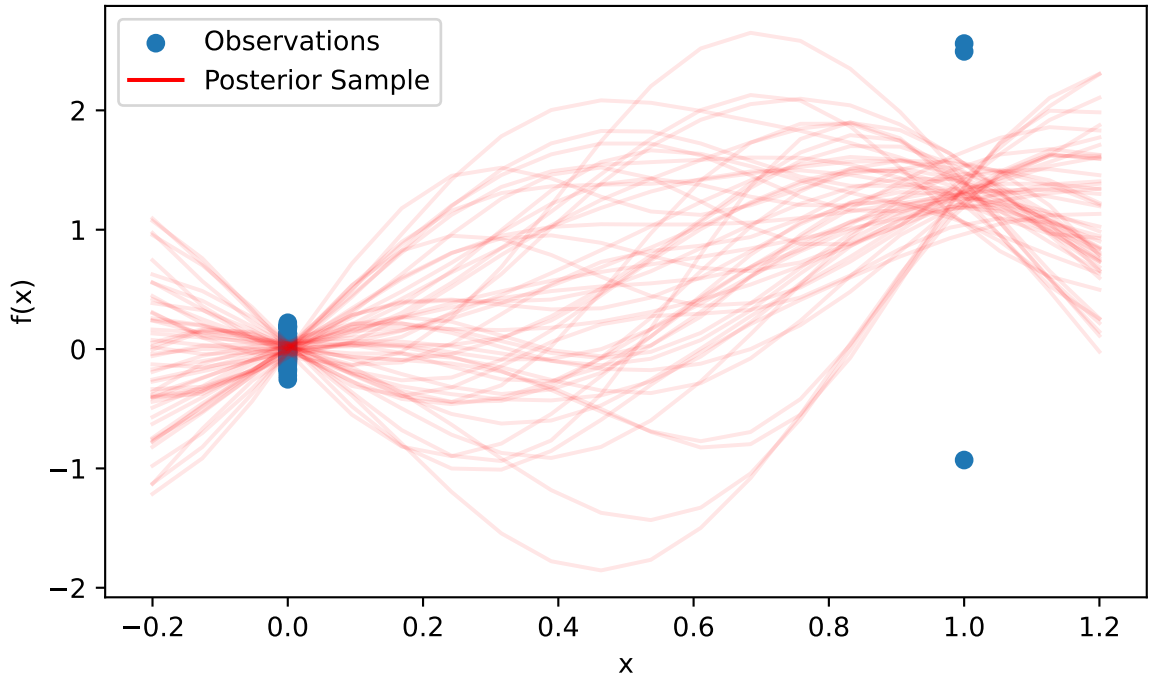
```
varying_variance(3, 3, 0.01, 1)
varying_variance(3, 10, 0.01, 1)
varying_variance(3, 100, 0.01, 1)
varying_variance(10, 3, 0.01, 1)
varying_variance(100, 3, 0.01, 1)
```




Observations
Posterior Sample
f(x)
x
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Observations
Posterior Sample
f(x)
x
3
2
1
0
−1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Observations
Posterior Sample
f(x)
x
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Observations
Posterior Sample
f(x)
x
2
1
0
−1
−2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2