Lab 2 Solution - GP Regresson in BoTorch

July 1, 2022

1 Lab 2 Solution: Gaussian process regression in BoTorch and gpytorch

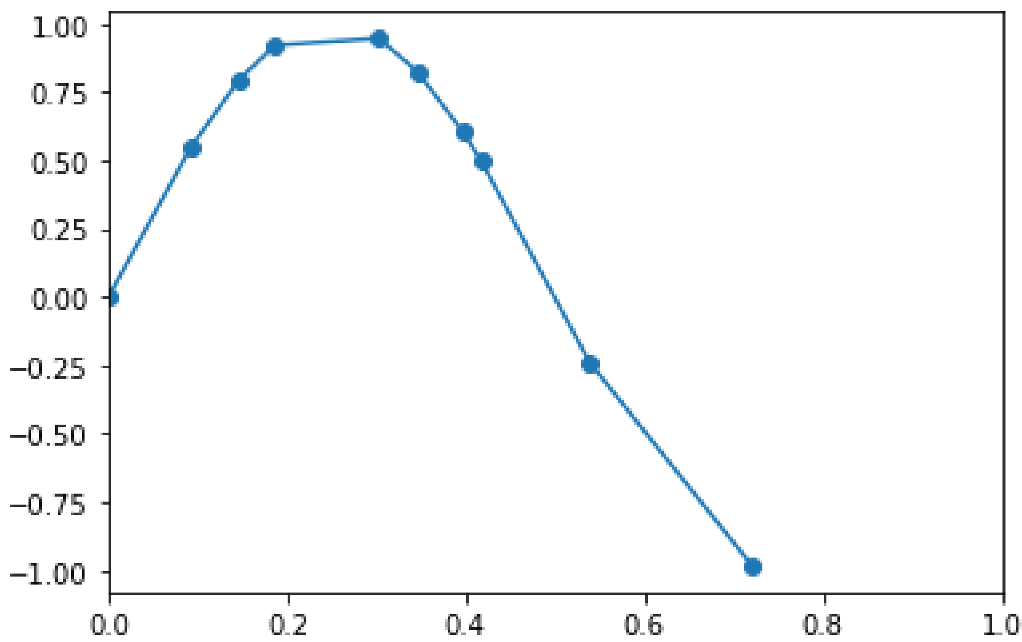
BoTorch is a package for Bayesian Optimization. Its support for Gaussian process regression comes from another package, gpytorch. Both of these packages are, in turn, based on pytorch, which is a framework often used for deep learning that supports fast operations on GPUs in python.

Here we'll use these packages to do the same inference that we did above, but more quickly and with more features. The format for this portion of the tutorial is based on the BoTorch tutorial, https://botorch.org/tutorials/fit_model_with_torch_optimizer, which in turn seems to be based on the gpytorch tutorial, https://github.com/cornellius-gp/gpytorch/blob/master/examples/01_Exact_GPs/Simple_GP_Regression.ipynb

[1]:
```
import torch # loading torch before matplotlib can be important
import matplotlib.pyplot as plt
import math
import numpy as np
import os.path
```

[2]:
```
filename = 'lab1_data.csv'
data = np.loadtxt(filename)
train_x = data[:,0] # First column of the data
train_y = data[:,1] # Second column of the data
plt.xlim(0,1)
plt.plot(train_x, train_y,'o-')
```

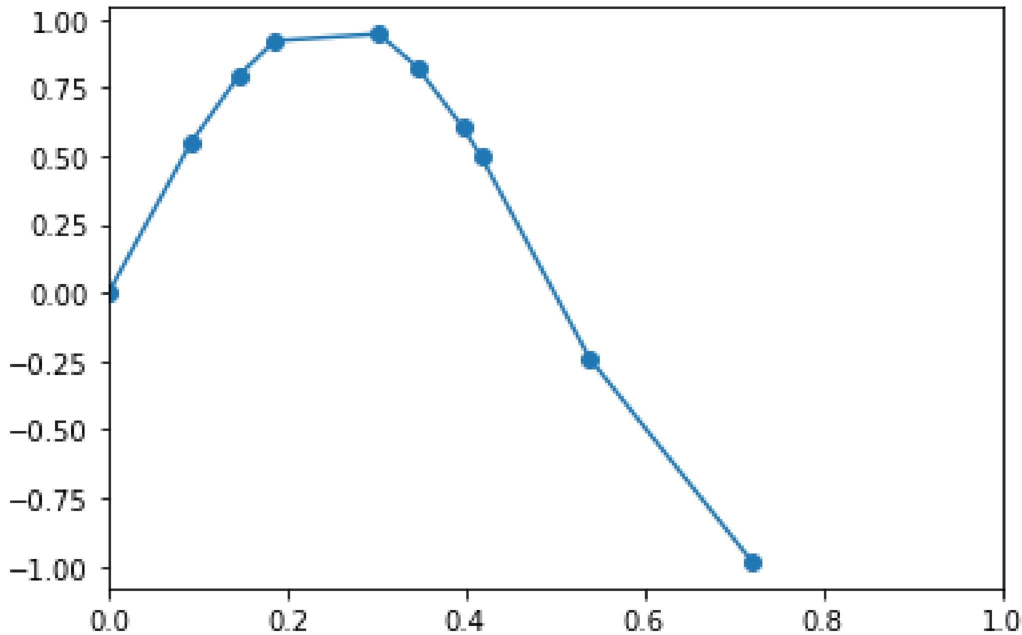
[2]:
```
[<matplotlib.lines.Line2D at 0x7ff790652e50>]
```



[3]:
```
# use a GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
dtype = torch.double
```

[4]:
```
# Read in data from a file.  Code copied from Lab 1.
filename = 'lab1_data.csv'

# If data doesn't exist, generate it
if ~os.path.exists(filename):
    np.random.seed(1)
    train_x = np.sort(np.random.rand(10)) # 10 points, uniformly distributed␣
 ↪between 0 and 1
    train_y = [math.sin(x * (2 * math.pi)) + 0.0 * np.random.randn() for x in␣
 ↪train_x]
    data = np.transpose([train_x,train_y])
    np.savetxt(filename,data)

# Read in data from a file.
data = np.loadtxt(filename)
train_x = data[:,0] # First column of the data
train_y = data[:,1] # Second column of the data
plt.xlim(0,1)
plt.plot(train_x, train_y,'o-')
```

[4]: `[<matplotlib.lines.Line2D at 0x7ff760b169d0>]`



1.0.1 Convert training data to torch format

[5]:
```
# Convert our training x and y data to torch format.
# To do this, we first convert to a torch tensor.
# Then we use unsqueeze to make the tensors multidimensional.
# Note that we are changing the datatype of train_x and train_y
train_x = torch.from_numpy(train_x).unsqueeze(1)
train_y = torch.from_numpy(train_y).unsqueeze(1)

# Outputting one of these tensors to show what they look like
train_x
```

[5]:
```
tensor([[1.1437e-04],
        [9.2339e-02],
        [1.4676e-01],
        [1.8626e-01],
        [3.0233e-01],
        [3.4556e-01],
        [3.9677e-01],
        [4.1702e-01],
        [5.3882e-01],
        [7.2032e-01]], dtype=torch.float64)
```

1.0.2 Train the GP regression model and plot the posterior

We will model the function using a `SingleTaskGP`, which by default uses a `GaussianLikelihood` and infers the unknown noise level.

The default optimizer for the `SingleTaskGP` is L-BFGS-B, which takes as input explicit bounds on the noise parameter. However, the `torch` optimizers don't support parameter bounds as input. To use the `torch` optimizers, then, we'll need to manually register a constraint on the noise level. When registering a constraint, the `softplus` transform is applied by default, enabling us to enforce a lower bound on the noise.

Note: Without manual registration, the model itself does not apply any constraints, due to the interaction between constraints and transforms. Although the `SingleTaskGP` constructor does in fact define a constraint, the constructor sets `transform=None`, which means that the constraint is not enforced. See the GPyTorch constraints module for additional information.

[6]:
```python
from botorch.models import SingleTaskGP
from gpytorch.constraints import GreaterThan
from gpytorch.likelihoods import FixedNoiseGaussianLikelihood

# Create a likelihood appropriate for noise-free observations.
# We will use a Gaussian likelihood with a very small variance␣
↪(noisefree_variance).
# We use a small strictly positive variance instead of zero variance
# to avoid numerical instabilities. If, instead, we set noisefree_variance to␣
↪0,
# then gpytorch would issue warning messages and round up to a small positive␣
↪number.
noisefree_variance = 0.0001
noises = torch.ones(len(train_y)) * noisefree_variance
noise_free_likelihood = FixedNoiseGaussianLikelihood(noise=noises)

# Create our GP model using the training data. By default, BoTorch uses a␣
↪constant mean and Matern kernel.
model = SingleTaskGP(train_x, train_y, likelihood = noise_free_likelihood)

# Another slightly faster way to do the same thing is:

# from botorch.models import FixedNoiseGP
# model = FixedNoiseGP(torch_train_x, torch_train_y, noises)

# If you want to allow BoTorch to learn the variance of homoscedastic Gaussian␣
↪noise, drop the likelihood argument as in
# model = SingleTaskGP(train_x, train_y, likelihood = noise_free_likelihood)

model
```

```
[6]: SingleTaskGP(
  (likelihood): FixedNoiseGaussianLikelihood(
    (noise_covar): FixedGaussianNoise()
  )
  (mean_module): ConstantMean()
  (covar_module): ScaleKernel(
    (base_kernel): MaternKernel(
      (lengthscale_prior): GammaPrior()
      (raw_lengthscale_constraint): Positive()
    )
    (outputscale_prior): GammaPrior()
    (raw_outputscale_constraint): Positive()
  )
)
```

You can see from the output above that the GP model has several components: - a likelihood, which describes the likelihood of our observation in terms of the value of the underlying function we want to estimate. We assume that our observations are normally distributed with a mean equal to the function's true value, and a very small variance. One can also assume that we observe the function without noise, but this creates some numerical instabilities in gpytorch. - a mean and covar module, which specifies the mean and kernel function respectively

The covar module has a constraint on its lengthscale saying that it must be positive, and also has a prior that is used when estimating it from data. For now, we will simply use a fixed length scale and so this constraint and prior won't matter for the moment.

Now we change the parameters used in the prior to match what we used in Lab 1 as a default. The constant for the mean because it is 0 by default. Then we can fit the model and see that BoTorch is doing the same thing that we did earlier.

```
[7]: # Because of the way this module is written, we have to construct a neural␣
 ↪network parameter
# from the desired value (0), rather than simply setting the constant to 0.
model.mean_module.constant = torch.nn.Parameter(torch.tensor(0.))

# gpytorch only supports these values for the nu parameter for the Matern␣
 ↪kernel: 1/2, 3/2, or 5/2
# If you want to use the value 2, this is supported through the RBF Kernel.
# (The RBF kernel is the Matern kernel with nu=2).
model.covar_module.base_kernel.nu = 1.5

# base_kernel.lengthscale is what we call length_scale in Lab 1 and what we␣
 ↪call alpha_1 in the slides.
# It determines the length scale of the output.
model.covar_module.base_kernel.lengthscale = 1.

# model.covar_module.outputscale is what we call alpha0 in our Lab 1 code.
model.covar_module.outputscale = 1.
```

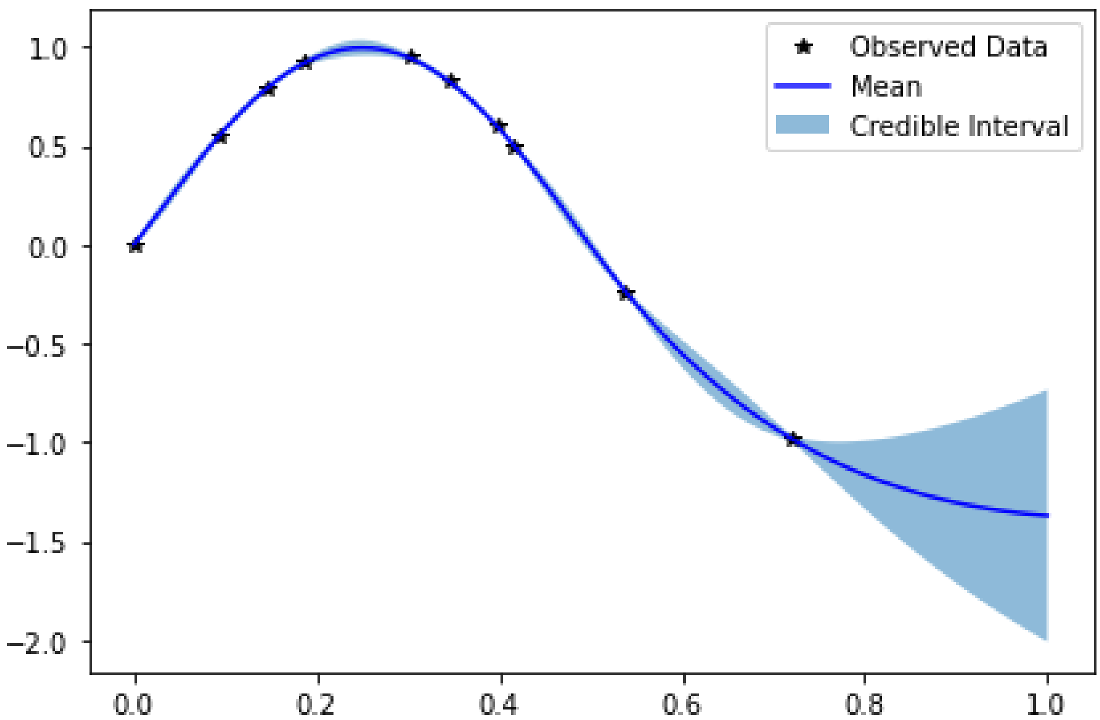
```python
# Train the GP model
model.eval()
```

[7]:
```
SingleTaskGP(
  (likelihood): FixedNoiseGaussianLikelihood(
    (noise_covar): FixedGaussianNoise()
  )
  (mean_module): ConstantMean()
  (covar_module): ScaleKernel(
    (base_kernel): MaternKernel(
      (lengthscale_prior): GammaPrior()
      (raw_lengthscale_constraint): Positive()
    )
    (outputscale_prior): GammaPrior()
    (raw_outputscale_constraint): Positive()
  )
)
```

[8]:
```python
def plot_posterior(model):

    # Initialize plot
    f, ax = plt.subplots(1, 1, figsize=(6, 4))
    # test model on 101 regular spaced points on the interval [0, 1]
    test_x = torch.linspace(0, 1, 101, dtype=dtype, device=device)

    with torch.no_grad(): # no need for gradients
        # compute posterior
        posterior = model.posterior(test_x)
        # Get upper and lower confidence bounds (2 standard deviations from the␣
↪mean)
        lower, upper = posterior.mvn.confidence_region()
        # Plot training points as black stars
        ax.plot(train_x.cpu().numpy(), train_y.cpu().numpy(), 'k*')
        # Plot posterior means as blue line
        ax.plot(test_x.cpu().numpy(), posterior.mean.cpu().numpy(), 'b')
        # Shade between the lower and upper confidence bounds
        ax.fill_between(test_x.cpu().numpy(), lower.cpu().numpy(), upper.cpu().
↪numpy(), alpha=0.5)

    ax.legend(['Observed Data', 'Mean', 'Credible Interval'])
    plt.tight_layout()

plot_posterior(model)
```

```
/usr/local/anaconda3/lib/python3.7/site-
packages/gpytorch/lazy/lazy_tensor.py:1810: UserWarning: torch.triangular_solve
```

```
is deprecated in favor of torch.linalg.solve_triangularand will be removed in a
future PyTorch release.
torch.linalg.solve_triangular has its arguments reversed and does not return a
copy of one of the inputs.
X = torch.triangular_solve(B, A).solution
should be replaced with
X = torch.linalg.solve_triangular(A, B). (Triggered internally at
/Users/runner/work/_temp/anaconda/conda-bld/pytorch_1656352443756/work/aten/src/
ATen/native/BatchLinearAlgebra.cpp:2189.)
  Linv = torch.triangular_solve(Eye, L, upper=False).solution
```



You should see that this plot matches very closely with the plot that we got in Lab 1 using the default parameters:

```
plot_prediction()
```

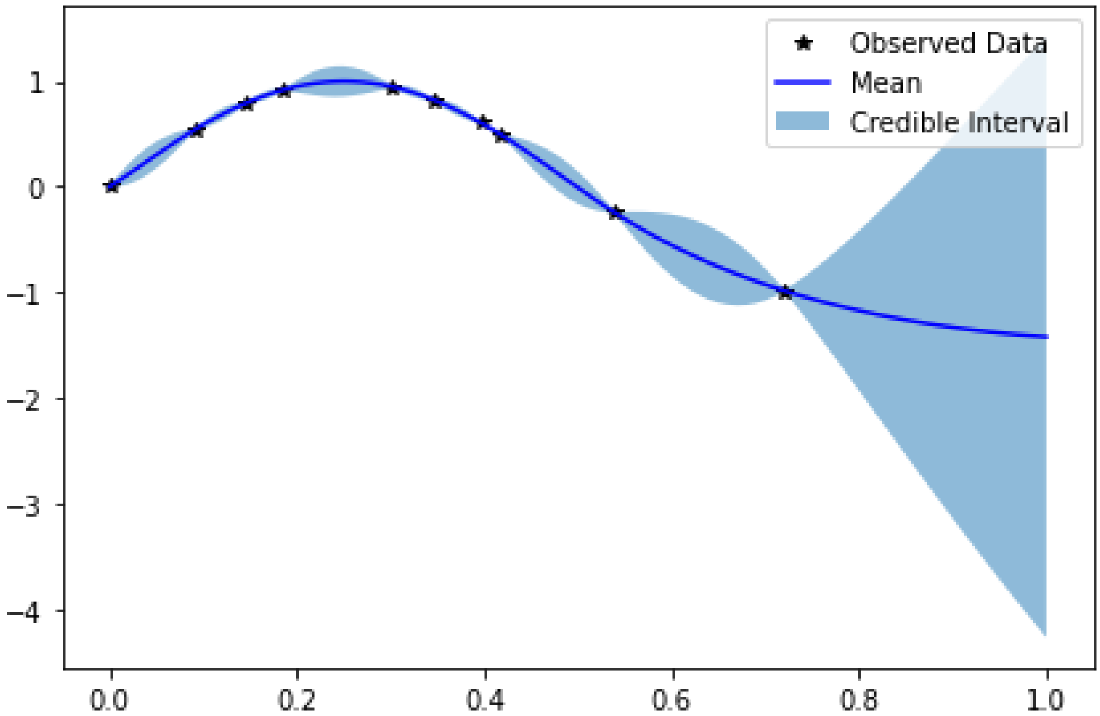
Exercise 1 Using BoTorch, reproduce the figure that you generated in Lab 1 with this code

```
plot_prediction(     mean = lambda x : -0.5,     kernel = lambda x1,x2 :
matern(x1,x2, alpha0=20, length_scales=1.0, nu = 1.5))
```

Exercise 1 Solution

[9]:
```
model = SingleTaskGP(train_x, train_y, likelihood = noise_free_likelihood)

model.mean_module.constant = torch.nn.Parameter(torch.tensor(-0.5))
```

```
model.covar_module.base_kernel.nu = 1.5
model.covar_module.base_kernel.lengthscale = 1.
model.covar_module.outputscale = 20.

# Train the GP model
model.eval()

plot_posterior(model)
```



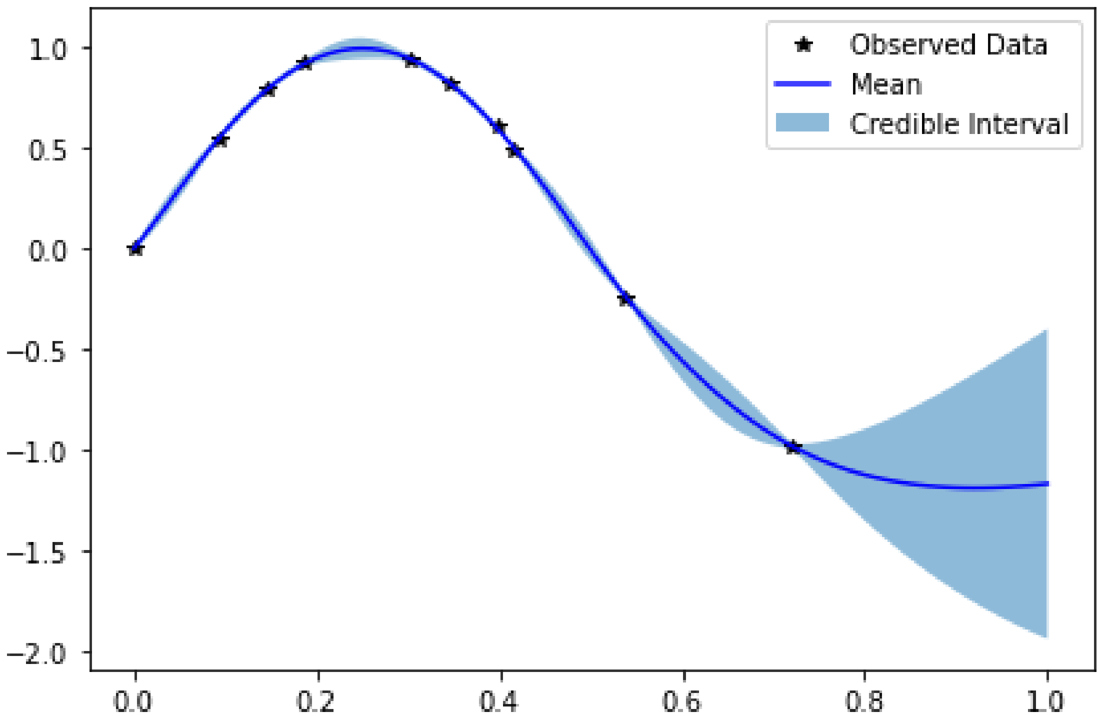
End Exercise 1

2 Choosing the GP hyperparameters

Here we show how to optimize the GP hyperparameters (the lengthscale, output scale, and constant mean of the prior — nu will remain fixed) based on the log marginal likelihood. Here we will show you how to do this manually, explaining all of the steps, and then below we'll show a function that does all of this for you.

2.0.1 Start with a fresh Gaussian process, that doesn't have the parameters we set above

[10]:
```
model = SingleTaskGP(train_x, train_y, likelihood = noise_free_likelihood)
model.covar_module.base_kernel.nu = 1.5
model.eval();

plot_posterior(model)
```



Define marginal log likelihood (mll)

[11]:
```
from gpytorch.mlls import ExactMarginalLogLikelihood

mll = ExactMarginalLogLikelihood(likelihood=model.likelihood, model=model)

# set mll and all submodules to the numeric data type and device (GPU vs. CPU)␣
↪specified at the start of
# our BoTorch code.
mll = mll.to(train_x)
```

Define optimizer and specify hyperparameters to optimize We will use stochastic gradient descent (`torch.optim.SGD`) to optimize the hyperparameters. In this example, we will use a simple fixed learning rate of 0.2, but in practice the learning rate may need to be adjusted. We will optimize over all parameters in the model, though one could pass a subset of the parameters to the model to tune only those while leaving the others fixed.

[12]:
```
from torch.optim import SGD

optimizer = SGD([{'params': model.parameters()}], lr=0.2)
```

The following code allows us to look at the names of the parameters in the GP regression model that will be optimized. Note that the outputscale and lengthscale are outputted with a "raw" prepended to them. gpytorch wants to ensure that certaint parameters (like the lengthscale and outputscale) satisfy constraints (specifially, that they are positive). To do this, it maintains a raw version of these parameters that can be any real number, and a transformed version that satisfies the constraint. For example, to satisfy a non-negativity constraint, one can have the transformed version be exp() applied to the raw parameter. The parameter nu is not included in the parameters that BoTorch optimizes.

[13]:
```
print('These are the parameters we will optimize over:')
for param_name, param in model.named_parameters():
    print(param_name)
```

```
These are the parameters we will optimize over:
mean_module.constant
covar_module.raw_outputscale
covar_module.base_kernel.raw_lengthscale
```

Optimize hyperparameters Now we are ready to write our optimization loop. We will perform 1500 epochs of stochastic gradient descent using our entire training set.

[14]:
```
NUM_EPOCHS = 1500

model.train()

for epoch in range(NUM_EPOCHS):
    # clear gradients
    optimizer.zero_grad()
    # forward pass through the model to obtain the output MultivariateNormal
    output = model(train_x)
    # Compute the negative marginal log likelihood
    loss = - mll(output, model.train_targets)
    # Use backward propagation to update the model's
    loss.backward()
    # print every 10 iterations
    if (epoch + 1) % 10 == 0:
        print(
            f"Epoch {epoch+1:>3}/{NUM_EPOCHS} - Negative Marginal Log␣
↪Likelihood : {loss.item():>4.3f} "
            f"lengthscale: {model.covar_module.base_kernel.lengthscale.item():
↪>4.3f} "
            f"constant: {model.mean_module.constant.item():>4.3f} "
```

```python
            f"outputscale: {model.covar_module.outputscale.item():>4.3f}"
        )
    optimizer.step()
```

```
Epoch  10/1500 - Negative Marginal Log Likelihood : -0.088 lengthscale: 0.515
constant: -0.184 outputscale: 0.836
Epoch  20/1500 - Negative Marginal Log Likelihood : -0.100 lengthscale: 0.529
constant: -0.305 outputscale: 0.893
Epoch  30/1500 - Negative Marginal Log Likelihood : -0.108 lengthscale: 0.543
constant: -0.390 outputscale: 0.944
Epoch  40/1500 - Negative Marginal Log Likelihood : -0.113 lengthscale: 0.554
constant: -0.452 outputscale: 0.990
Epoch  50/1500 - Negative Marginal Log Likelihood : -0.117 lengthscale: 0.564
constant: -0.500 outputscale: 1.033
Epoch  60/1500 - Negative Marginal Log Likelihood : -0.119 lengthscale: 0.573
constant: -0.537 outputscale: 1.072
Epoch  70/1500 - Negative Marginal Log Likelihood : -0.122 lengthscale: 0.581
constant: -0.567 outputscale: 1.109
Epoch  80/1500 - Negative Marginal Log Likelihood : -0.123 lengthscale: 0.588
constant: -0.591 outputscale: 1.143
Epoch  90/1500 - Negative Marginal Log Likelihood : -0.125 lengthscale: 0.594
constant: -0.611 outputscale: 1.174
Epoch 100/1500 - Negative Marginal Log Likelihood : -0.126 lengthscale: 0.600
constant: -0.628 outputscale: 1.203
Epoch 110/1500 - Negative Marginal Log Likelihood : -0.127 lengthscale: 0.605
constant: -0.643 outputscale: 1.230
Epoch 120/1500 - Negative Marginal Log Likelihood : -0.127 lengthscale: 0.609
constant: -0.656 outputscale: 1.255
Epoch 130/1500 - Negative Marginal Log Likelihood : -0.128 lengthscale: 0.613
constant: -0.667 outputscale: 1.278
Epoch 140/1500 - Negative Marginal Log Likelihood : -0.129 lengthscale: 0.617
constant: -0.677 outputscale: 1.300
Epoch 150/1500 - Negative Marginal Log Likelihood : -0.129 lengthscale: 0.621
constant: -0.686 outputscale: 1.320
Epoch 160/1500 - Negative Marginal Log Likelihood : -0.129 lengthscale: 0.624
constant: -0.694 outputscale: 1.339
Epoch 170/1500 - Negative Marginal Log Likelihood : -0.130 lengthscale: 0.627
constant: -0.701 outputscale: 1.357
Epoch 180/1500 - Negative Marginal Log Likelihood : -0.130 lengthscale: 0.630
constant: -0.707 outputscale: 1.373
Epoch 190/1500 - Negative Marginal Log Likelihood : -0.130 lengthscale: 0.632
constant: -0.713 outputscale: 1.388
Epoch 200/1500 - Negative Marginal Log Likelihood : -0.130 lengthscale: 0.635
constant: -0.719 outputscale: 1.402
Epoch 210/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.637
constant: -0.724 outputscale: 1.416
Epoch 220/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.639
```

```
constant: -0.728 outputscale: 1.428
Epoch 230/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.641
constant: -0.733 outputscale: 1.440
Epoch 240/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.643
constant: -0.736 outputscale: 1.450
Epoch 250/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.644
constant: -0.740 outputscale: 1.461
Epoch 260/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.646
constant: -0.743 outputscale: 1.470
Epoch 270/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.647
constant: -0.747 outputscale: 1.479
Epoch 280/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.648
constant: -0.749 outputscale: 1.488
Epoch 290/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.650
constant: -0.752 outputscale: 1.495
Epoch 300/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.651
constant: -0.755 outputscale: 1.503
Epoch 310/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.652
constant: -0.757 outputscale: 1.510
Epoch 320/1500 - Negative Marginal Log Likelihood : -0.131 lengthscale: 0.653
constant: -0.759 outputscale: 1.516
Epoch 330/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.654
constant: -0.761 outputscale: 1.522
Epoch 340/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.655
constant: -0.763 outputscale: 1.528
Epoch 350/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.656
constant: -0.765 outputscale: 1.534
Epoch 360/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.656
constant: -0.767 outputscale: 1.539
Epoch 370/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.657
constant: -0.768 outputscale: 1.543
Epoch 380/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.658
constant: -0.770 outputscale: 1.548
Epoch 390/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.658
constant: -0.771 outputscale: 1.552
Epoch 400/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.659
constant: -0.772 outputscale: 1.556
Epoch 410/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.660
constant: -0.774 outputscale: 1.560
Epoch 420/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.660
constant: -0.775 outputscale: 1.563
Epoch 430/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.661
constant: -0.776 outputscale: 1.567
Epoch 440/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.661
constant: -0.777 outputscale: 1.570
Epoch 450/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.662
constant: -0.778 outputscale: 1.573
Epoch 460/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.662
```

```
constant: -0.779 outputscale: 1.576
Epoch 470/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.662
constant: -0.779 outputscale: 1.578
Epoch 480/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.663
constant: -0.780 outputscale: 1.581
Epoch 490/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.663
constant: -0.781 outputscale: 1.583
Epoch 500/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.663
constant: -0.782 outputscale: 1.585
Epoch 510/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.664
constant: -0.782 outputscale: 1.587
Epoch 520/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.664
constant: -0.783 outputscale: 1.589
Epoch 530/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.664
constant: -0.784 outputscale: 1.591
Epoch 540/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.664
constant: -0.784 outputscale: 1.593
Epoch 550/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.665
constant: -0.785 outputscale: 1.594
Epoch 560/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.665
constant: -0.785 outputscale: 1.596
Epoch 570/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.665
constant: -0.786 outputscale: 1.597
Epoch 580/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.665
constant: -0.786 outputscale: 1.599
Epoch 590/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.786 outputscale: 1.600
Epoch 600/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.787 outputscale: 1.601
Epoch 610/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.787 outputscale: 1.602
Epoch 620/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.788 outputscale: 1.603
Epoch 630/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.788 outputscale: 1.604
Epoch 640/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.788 outputscale: 1.605
Epoch 650/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.666
constant: -0.788 outputscale: 1.606
Epoch 660/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.789 outputscale: 1.607
Epoch 670/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.789 outputscale: 1.608
Epoch 680/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.789 outputscale: 1.609
Epoch 690/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.789 outputscale: 1.609
Epoch 700/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
```

```
constant: -0.790 outputscale: 1.610
Epoch 710/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.790 outputscale: 1.611
Epoch 720/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.790 outputscale: 1.611
Epoch 730/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.790 outputscale: 1.612
Epoch 740/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.790 outputscale: 1.613
Epoch 750/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.667
constant: -0.791 outputscale: 1.613
Epoch 760/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.614
Epoch 770/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.614
Epoch 780/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.614
Epoch 790/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.615
Epoch 800/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.615
Epoch 810/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.616
Epoch 820/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.791 outputscale: 1.616
Epoch 830/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.616
Epoch 840/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.617
Epoch 850/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.617
Epoch 860/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.617
Epoch 870/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.617
Epoch 880/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.618
Epoch 890/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.618
Epoch 900/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.618
Epoch 910/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.618
Epoch 920/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.618
Epoch 930/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.619
Epoch 940/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
```

```
constant: -0.792 outputscale: 1.619
Epoch 950/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.619
Epoch 960/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.792 outputscale: 1.619
Epoch 970/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.619
Epoch 980/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.619
Epoch 990/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.619
Epoch 1000/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.620
Epoch 1010/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.620
Epoch 1020/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.620
Epoch 1030/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.620
Epoch 1040/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.668
constant: -0.793 outputscale: 1.620
Epoch 1050/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1060/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1070/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1080/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1090/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1100/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.620
Epoch 1110/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1120/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1130/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1140/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1150/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1160/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1170/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1180/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
```

```
constant: -0.793 outputscale: 1.621
Epoch 1190/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1200/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1210/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1220/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1230/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1240/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1250/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1260/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1270/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1280/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1290/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1300/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1310/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1320/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1330/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1340/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1350/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1360/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1370/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1380/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1390/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1400/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1410/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1420/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
```

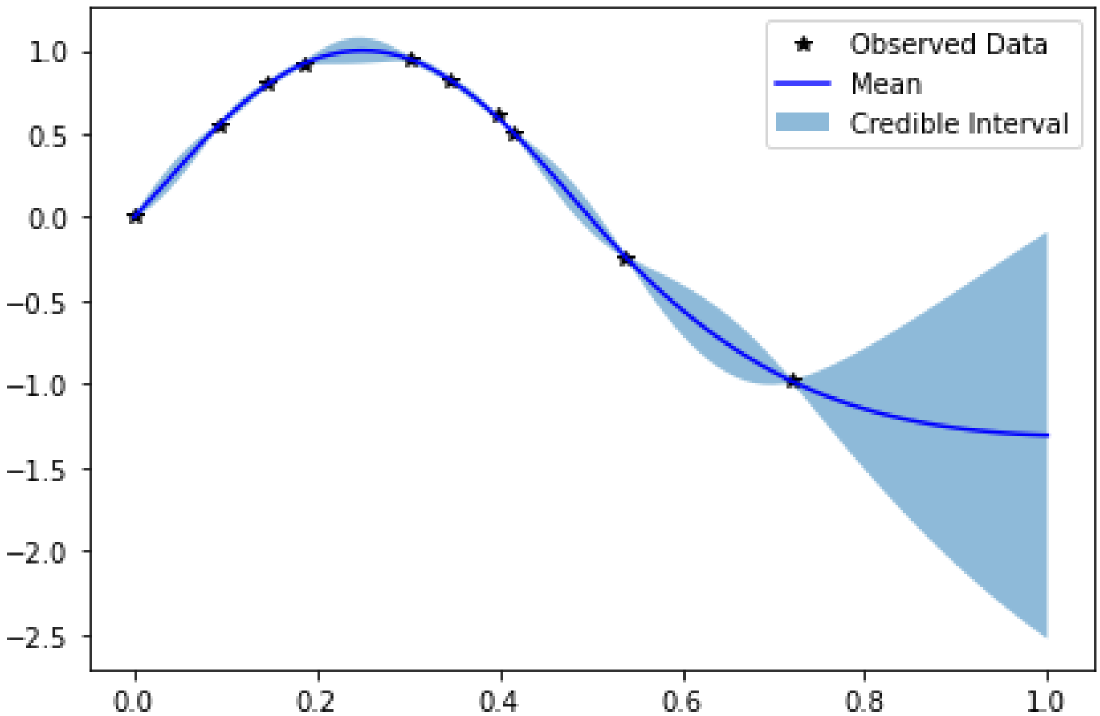
```
constant: -0.793 outputscale: 1.621
Epoch 1430/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1440/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1450/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.621
Epoch 1460/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.622
Epoch 1470/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.622
Epoch 1480/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.622
Epoch 1490/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.622
Epoch 1500/1500 - Negative Marginal Log Likelihood : -0.132 lengthscale: 0.669
constant: -0.793 outputscale: 1.622
```

[15]:
```
# set model (and likelihood)
model.eval()
```

[15]:
```
SingleTaskGP(
  (likelihood): FixedNoiseGaussianLikelihood(
    (noise_covar): FixedGaussianNoise()
  )
  (mean_module): ConstantMean()
  (covar_module): ScaleKernel(
    (base_kernel): MaternKernel(
      (lengthscale_prior): GammaPrior()
      (raw_lengthscale_constraint): Positive()
      (distance_module): Distance()
    )
    (outputscale_prior): GammaPrior()
    (raw_outputscale_constraint): Positive()
  )
)
```

[16]:
```
plot_posterior(model)
print(
    f"lengthscale: {model.covar_module.base_kernel.lengthscale.item():>4.3f} "
    f"constant: {model.mean_module.constant.item():>4.3f} "
    f"outputscale: {model.covar_module.outputscale.item():>4.3f} "
    f"nu: {model.covar_module.base_kernel.nu:>4.1f}"
)
```

```
lengthscale: 0.669 constant: -0.793 outputscale: 1.622 nu:  1.5
```



2.0.2 Here is a faster way to do the same thing

This uses a BoTorch helper function, fit_gpytorch_model, that isn't in gpytorch

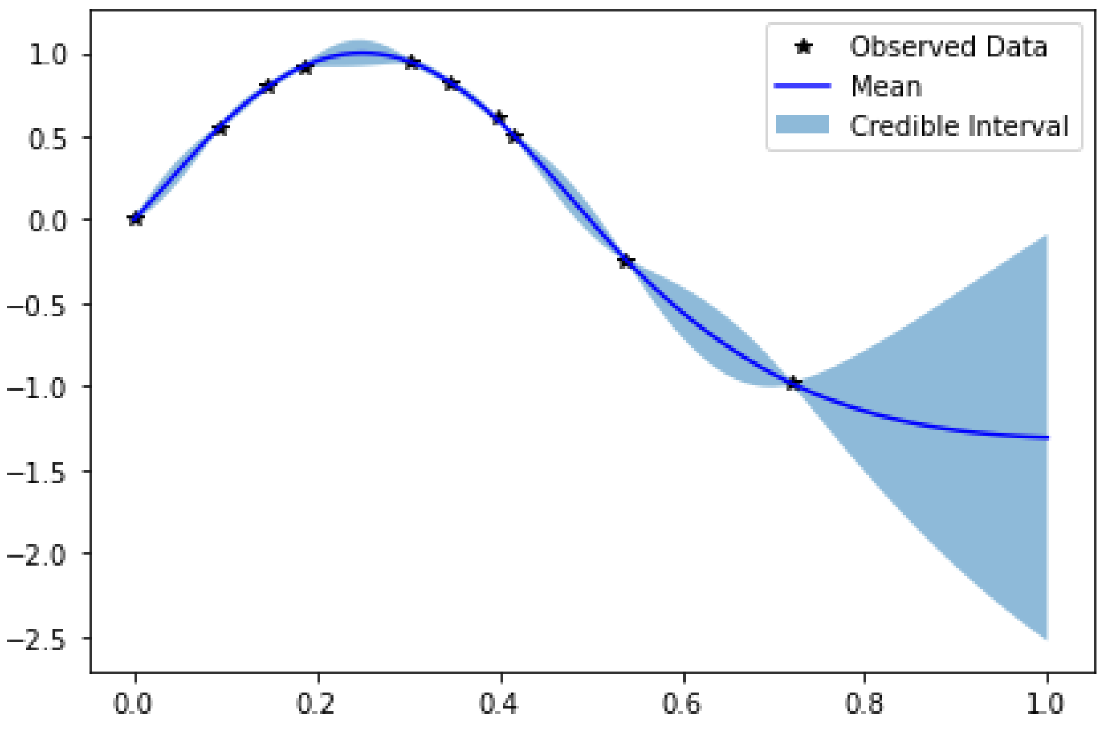
[17]:
```
from botorch.fit import fit_gpytorch_model
model = SingleTaskGP(train_x, train_y, likelihood =␣
 ↪FixedNoiseGaussianLikelihood(noise=noises))
model.covar_module.base_kernel.nu = 1.5
mll = ExactMarginalLogLikelihood(model.likelihood, model)
fit_gpytorch_model(mll) # This chooses the hyperparameters to maximize the log␣
 ↪marginal likelihood
```

[17]:
```
ExactMarginalLogLikelihood(
  (likelihood): FixedNoiseGaussianLikelihood(
    (noise_covar): FixedGaussianNoise()
  )
  (model): SingleTaskGP(
    (likelihood): FixedNoiseGaussianLikelihood(
      (noise_covar): FixedGaussianNoise()
    )
    (mean_module): ConstantMean()
    (covar_module): ScaleKernel(
      (base_kernel): MaternKernel(
        (lengthscale_prior): GammaPrior()
```

```
          (raw_lengthscale_constraint): Positive()
          (distance_module): Distance()
        )
        (outputscale_prior): GammaPrior()
        (raw_outputscale_constraint): Positive()
      )
    )
  )
```

[18]:
```
plot_posterior(model)
print(
    f"lengthscale: {model.covar_module.base_kernel.lengthscale.item():>4.3f} "
    f"constant: {model.mean_module.constant.item():>4.3f} "
    f"outputscale: {model.covar_module.outputscale.item():>4.3f} "
    f"nu: {model.covar_module.base_kernel.nu:>4.1f}"
)
```

```
lengthscale: 0.669 constant: -0.793 outputscale: 1.622 nu:  1.5
```



3 Cross-validation

The following cross-validation plots are for checking model fit.

[19]:
```
# Helper function for cross-validation
def remove(T,i):
    # Remove the ith component from the 1-dimensional tensor T
    assert(i<len(T))
    return torch.cat([T[:i], T[i+1:]])

# Check that we handle the two corner cases where i is 0 or len-1
assert(len(remove(train_x,0))==9)
assert(len(remove(train_x,len(train_x)-1))==9)
assert(len(remove(train_x,5))==9)
```

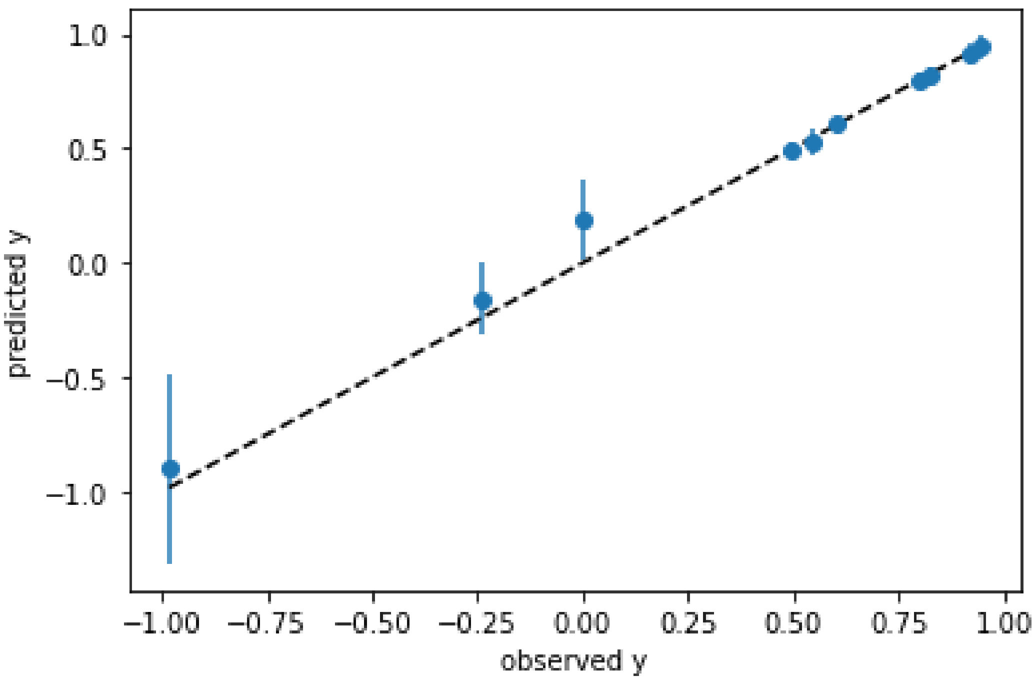
[20]:
```
remove(train_y,1)
```

[20]:
```
tensor([[ 7.1864e-04],
        [ 7.9687e-01],
        [ 9.2087e-01],
        [ 9.4643e-01],
        [ 8.2510e-01],
        [ 6.0409e-01],
        [ 4.9807e-01],
        [-2.4148e-01],
        [-9.8267e-01]], dtype=torch.float64)
```

[21]:
```
def cross_validation(train_x,train_y,nu=1.5):
    loo_mean = []
    loo_sdev = []

    for i in range(len(train_x)):

        # Remove the ith datapoint from the training set
        loo_train_x = remove(train_x,i)
        loo_train_y = remove(train_y,i)
        noisefree_variance = 0.0001
        noises = torch.ones(len(loo_train_y)) * noisefree_variance

        model = SingleTaskGP(loo_train_x, loo_train_y, likelihood =␣
 ↪FixedNoiseGaussianLikelihood(noise=noises))
        model.covar_module.base_kernel.nu = nu
        mll = ExactMarginalLogLikelihood(model.likelihood, model)
        fit_gpytorch_model(mll)

        posterior = model.posterior(train_x[i].unsqueeze(0))
        # the posterior mean and variance are 2-d 1x1 tensors.
        # We just need to pull out the single number inside each, which is at␣
 ↪index [0][0]
        m = posterior.mean.cpu().detach().numpy()[0][0]
        v = posterior.variance.cpu().detach().numpy()[0][0]
```

```
        loo_mean.append(m)
        loo_sdev.append(np.sqrt(v))

    fig, ax = plt.subplots()
    train_y_np = train_y.cpu().numpy()[:,0]

    ax.errorbar(train_y_np,loo_mean,loo_sdev,fmt='o')
    ax.
↪plot([min(train_y_np),max(train_y_np)],[min(train_y_np),max(train_y_np)],'k--')
    plt.xlabel('observed y')
    plt.ylabel('predicted y')
```

[22]:
```
cross_validation(train_x,train_y)
```



We want to see that the error bar crosses the dashed line for most of the datapoints (95% of the datapoints), and when it misses it shouldn't miss by too much. This plot looks pretty good.

Exercise 2 Using the function below (which is -1 times a standard test function called Hartmann6), and the 50 training points generated below (uniformly over the unit cube), fit GP regression using a Matern kernel with nu=.5 and no noise in the observations, Choose the hyperparameters by maximizing the log marginal likelihood.

Then plot a cross-validation plot. You'll see that it looks ok, but some errobars are far from touching the line. Based on this plot we might want consider another kernel or transforming the objective.

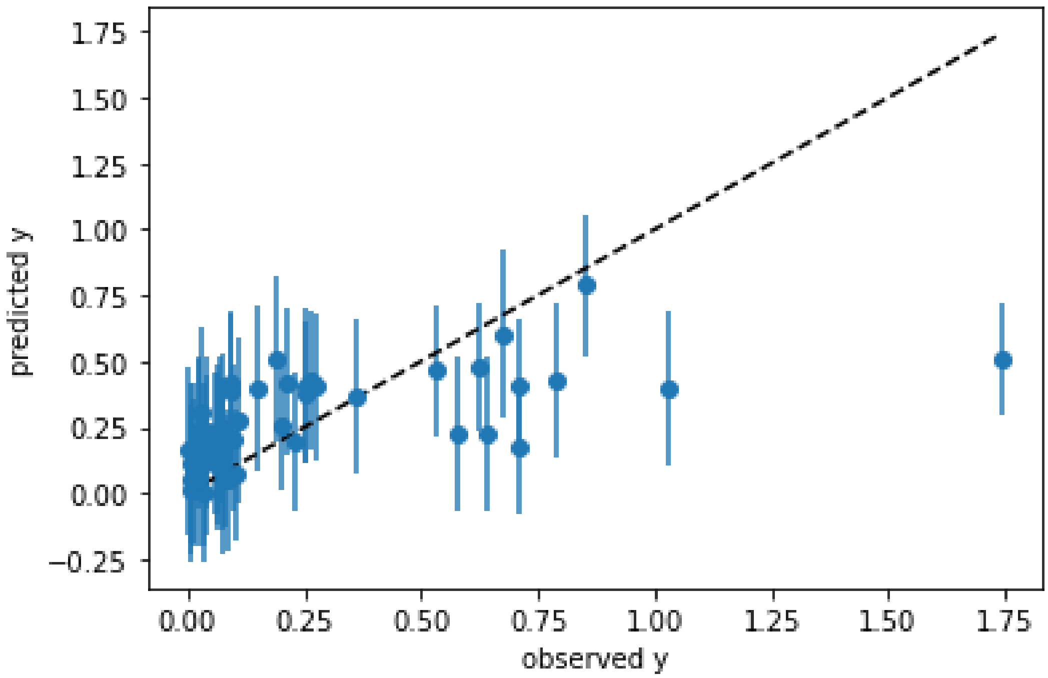
[23]:
```
from botorch.test_functions import Hartmann
f = Hartmann(negate=True)

np.random.seed(1)
train_x = torch.rand(50, 6, device=device, dtype=dtype)
train_y = f(train_x).unsqueeze(-1)  # add output dimension
```

Exercise 2 Solution

[24]:
```
noisefree_variance = 0.0001
noises = torch.ones(len(train_y)) * noisefree_variance
model = SingleTaskGP(train_x, train_y, likelihood =␣
 ↪FixedNoiseGaussianLikelihood(noise=noises))
model.covar_module.base_kernel.nu = .5
mll = ExactMarginalLogLikelihood(model.likelihood, model)
fit_gpytorch_model(mll)
```

[24]:
```
ExactMarginalLogLikelihood(
  (likelihood): FixedNoiseGaussianLikelihood(
    (noise_covar): FixedGaussianNoise()
  )
  (model): SingleTaskGP(
    (likelihood): FixedNoiseGaussianLikelihood(
      (noise_covar): FixedGaussianNoise()
    )
    (mean_module): ConstantMean()
    (covar_module): ScaleKernel(
      (base_kernel): MaternKernel(
        (lengthscale_prior): GammaPrior()
        (raw_lengthscale_constraint): Positive()
        (distance_module): Distance()
      )
      (outputscale_prior): GammaPrior()
      (raw_outputscale_constraint): Positive()
    )
  )
)
```

[25]:
```
cross_validation(train_x,train_y)
```


1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
predicted y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
observed y

[ ]: