02_gp_2d_inference

January 15, 2025

1 Bivariate Gaussian Process Regression

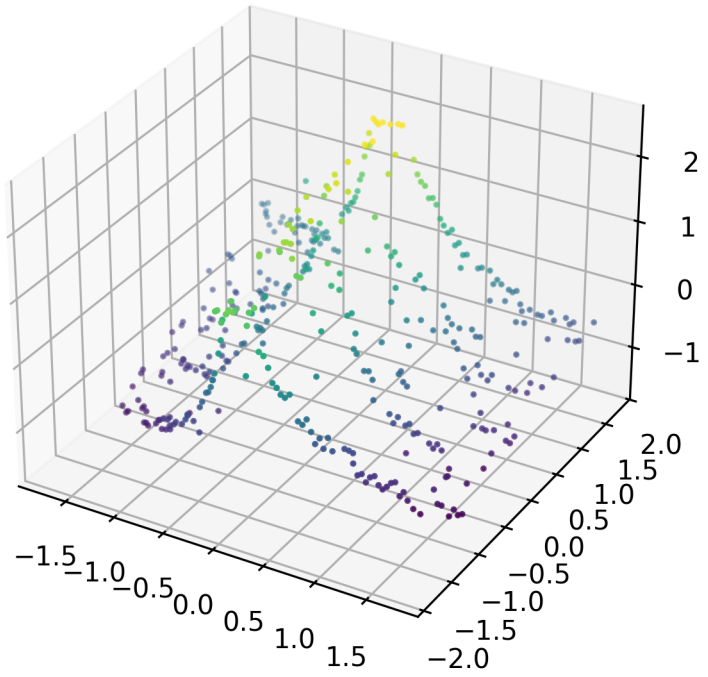
1.1 Required Modules

[1]:
```
import sys
sys.path.append("..")

import torch
import pyro
import pyro.contrib.gp as gp
import pyro.infer
import pyro.optim

import pyro.distributions as dist
from pyro.infer import Trace_ELBO
pyro.set_rng_seed(0)
```

```
/home/ale/pythonEnv/tut/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1.2 Data Retrieval and Inspection

[2]:
```
from src.data import read_data
from src.view import view_data_2D

# X_train, y_train = read_data("../point_cloud_reduced.xyz")
X_train, y_train = read_data("../data/point_cloud.xyz")
view_data_2D(X_train, y_train)
```



This dataset regards … DESCRIBE EXAMPLE

1.3 GP Regression Setting

$$y(x) = f(x) + \epsilon \qquad f \sim \mathrm{GP}(m(x), k(x)) \qquad \epsilon \sim \mathcal{N}(0, \sigma_n^2)$$

1.4 Initialising the Regression

[3]:
```
# initial guesses of kernel's parameters
var = torch.tensor(1.0)
length = torch.tensor((1.0, 1.0))
noise = torch.tensor(0.1)

# load RBF kernel and GP regression
kernel = gp.kernels.RBF(input_dim=2, variance=var, lengthscale=length)
gpr = gp.models.GPRegression(X_train, y_train, kernel, noise=noise)
```

1.5 Solving the Regression Problem

From a computational perspective, `pyro-ppl` seeks to maximise this loss function to accomplish regression:

$$\mathrm{ELBO} = \mathbb{E}_{\mathbb{Q}_\lambda[\theta]}\left[\log \mathbb{P}[\mathbf{x}|\theta] - \log \mathbb{Q}_\lambda[\theta]\right]$$

The optimisers (`Adam`) requires a learning rate to regulate the rate of the gradient evaluation of

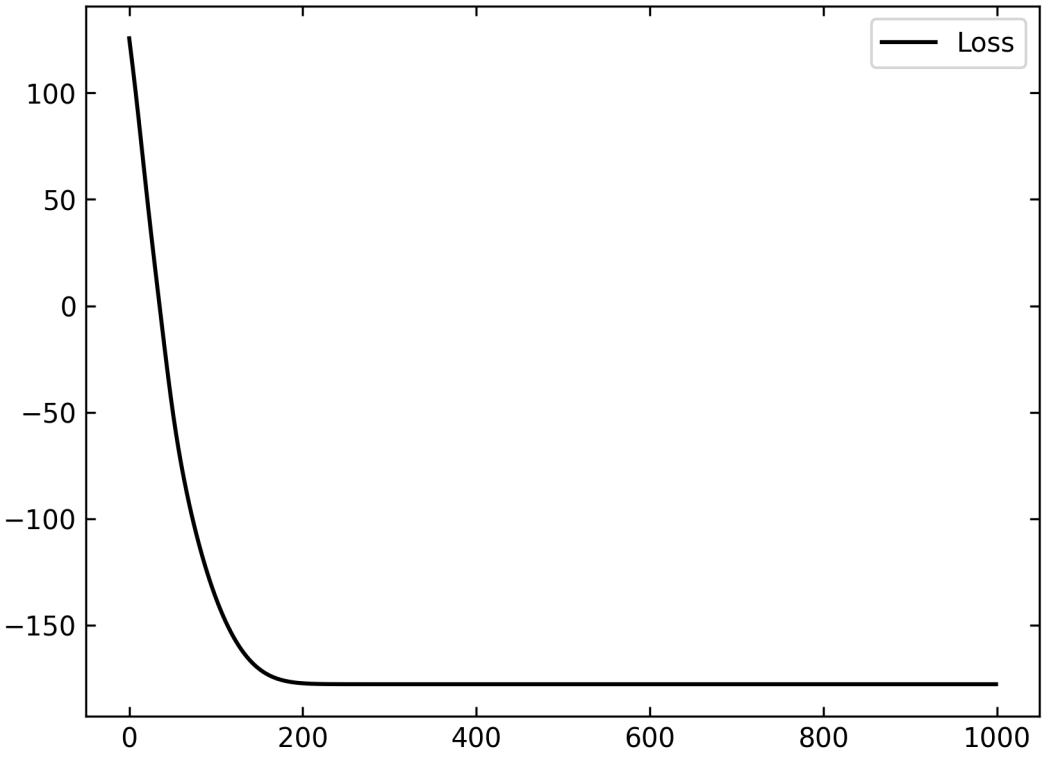
the loss function. Specifically, the loss function implemented by `pyro-ppl` is `Trace_ELBO`. We also need to specify how many `steps` should last the training. Finally, the training is handled the an helper function `train_helper`, which wraps `pyro-ppl` utilities.

[4]:
```
# load optimiser and loss function
learning_rate = 0.01
optimiser = pyro.optim.Adam({"lr": learning_rate})
loss_fn = Trace_ELBO()
steps = 1000
```

[5]:
```
from src.view import inspect_loss
from src.inference import train_helper
# train and inspect loss function
loss = train_helper(gpr, optimiser, loss_fn, steps)
inspect_loss(loss)
```

```
--- Step 0 - Loss: 125.65497890092195
* Variance: 1.0100501775741577
* Lengthscale: tensor([0.9900, 1.0101], grad_fn=<AddBackward0>)
* Noise: 0.09900498390197754

--- Step 100 - Loss: -137.31452267905632
* Variance: 1.616642713546753
* Lengthscale: tensor([0.4794, 2.9148], grad_fn=<AddBackward0>)
* Noise: 0.03274150565266609

--- Step 200 - Loss: -177.25498432239988
* Variance: 1.3321934938430786
* Lengthscale: tensor([0.4053, 4.8909], grad_fn=<AddBackward0>)
* Noise: 0.01844858191907406

--- Step 300 - Loss: -177.6921562407274
* Variance: 1.329177737236023
* Lengthscale: tensor([0.3979, 5.0931], grad_fn=<AddBackward0>)
* Noise: 0.017371196299791336

--- Step 400 - Loss: -177.69241324184065
* Variance: 1.3359442949295044
* Lengthscale: tensor([0.3980, 5.1047], grad_fn=<AddBackward0>)
* Noise: 0.01736351102590561

--- Step 500 - Loss: -177.69250425307905
* Variance: 1.340414047241211
* Lengthscale: tensor([0.3981, 5.1123], grad_fn=<AddBackward0>)
* Noise: 0.017363859340548515

--- Step 600 - Loss: -177.69253398653802
* Variance: 1.3431421518325806
```

```
* Lengthscale: tensor([0.3982, 5.1170], grad_fn=<AddBackward0>)
* Noise: 0.01736401580274105

--- Step 700 - Loss: -177.69254207197662
* Variance: 1.3446455001831055
* Lengthscale: tensor([0.3982, 5.1195], grad_fn=<AddBackward0>)
* Noise: 0.01736411638557911

--- Step 800 - Loss: -177.6925439084505
* Variance: 1.3453978300094604
* Lengthscale: tensor([0.3982, 5.1208], grad_fn=<AddBackward0>)
* Noise: 0.01736416481435299

--- Step 900 - Loss: -177.69254425678457
* Variance: 1.3457393646240234
* Lengthscale: tensor([0.3983, 5.1214], grad_fn=<AddBackward0>)
* Noise: 0.01736419089138508

---------- Summary ----------
* Variance: 1.3458789587020874
* Lengthscale: tensor([0.3983, 5.1216], grad_fn=<AddBackward0>)
* Noise: 0.01736420765519142
* Elaspsed time: 6.79 s
```



1.6 Making Predictions

Conditioning to the training data:

$$\mathbb{E}[x_*] = k(x_*, x)[k(x, x) + \sigma_n^2 \mathbf{I}]^{-1}\mathbf{y}$$

$$\mathbb{V}[x_*] = k(x_*, x_*) - k(x_*, x)[k(x, x) + \sigma_n^2 \mathbf{I}]^{-1}k(x, x_*)$$

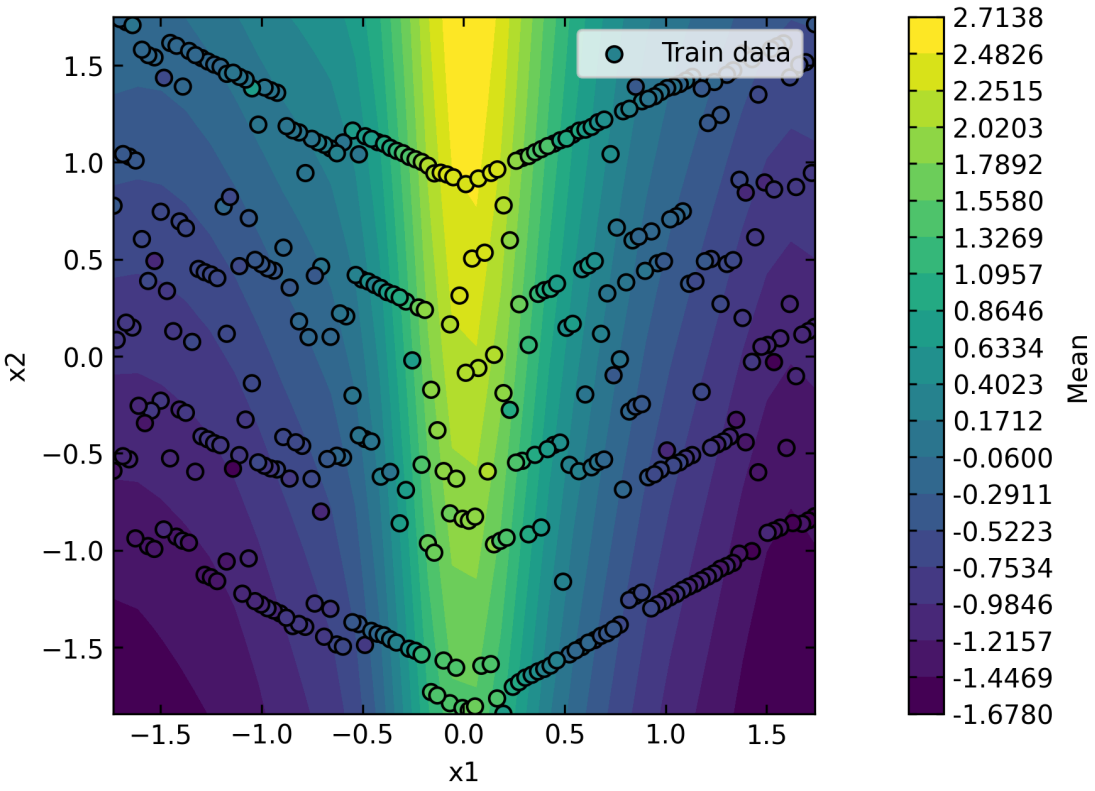
These operations are automatically handled by `forward` method of `gpr` (see `pyro-ppl` documentation).

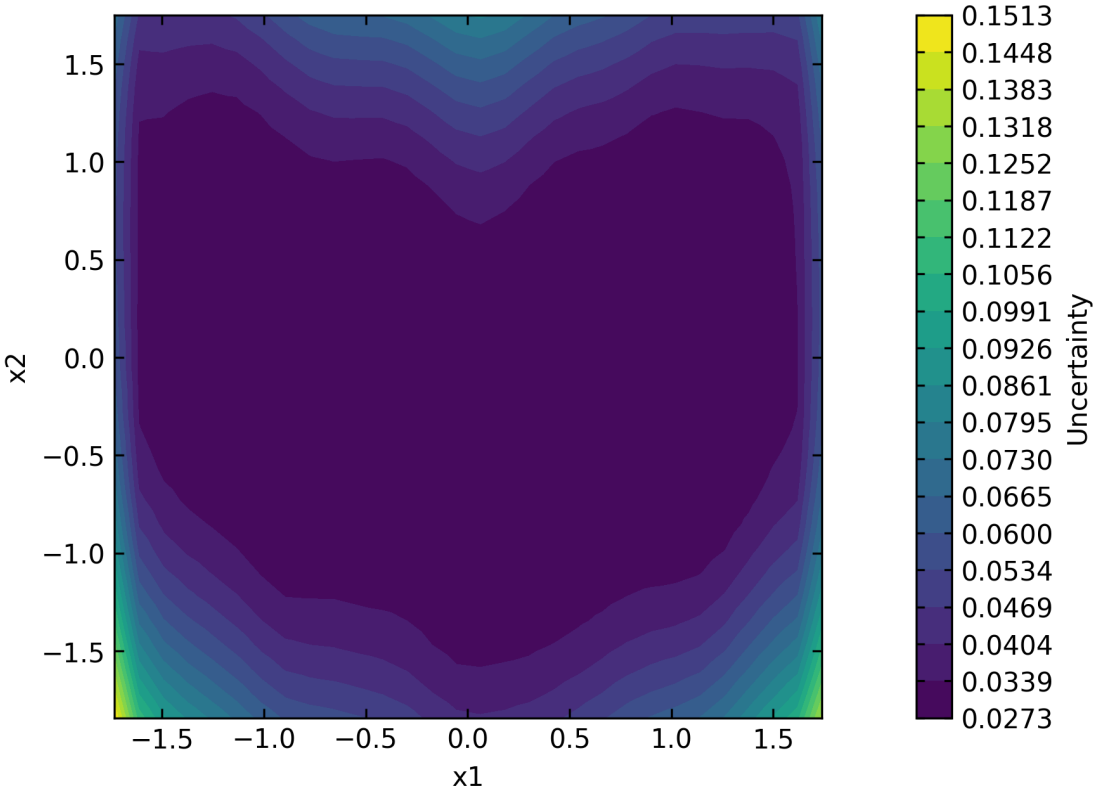
To evaluate the above, we make an *grid* of test data, condition it to the training data, and finally inspect the result.

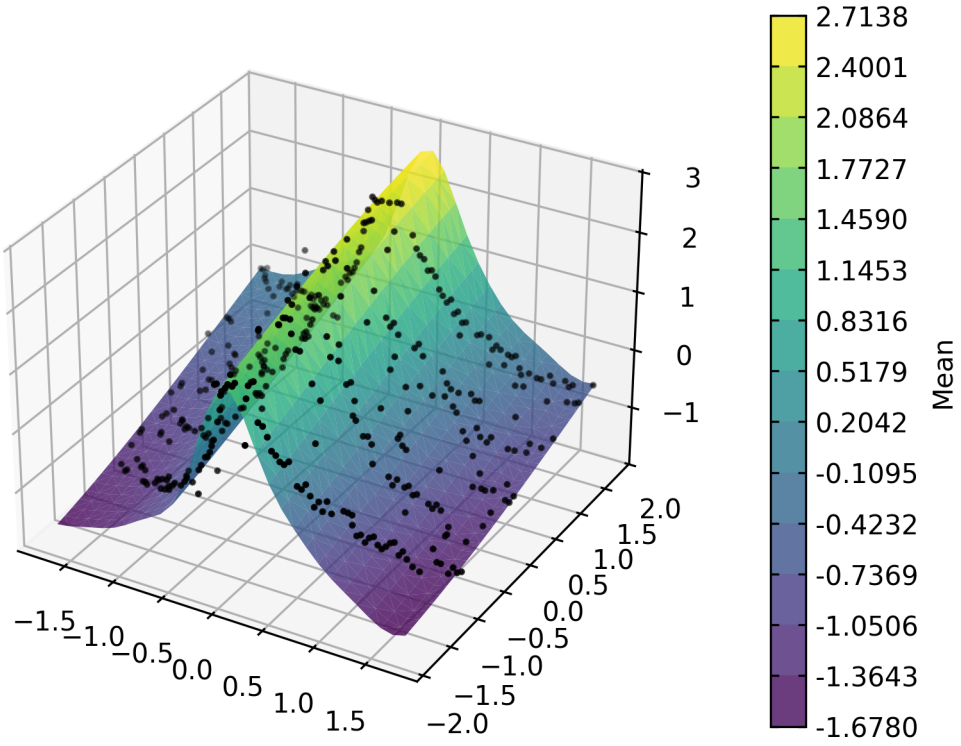
1.7 Graphical Inspection

[6]:
```
# grid test data
from src.data import grid_factory
X_test = grid_factory(X_train[:, 0], X_train[:, 1])

# conditioning
y_mean, y_cov = gpr(X_test, full_cov=True)
y_std = y_cov.diag().sqrt()
```

[7]:
```
from src.view import contour_and_scatter
from src.view import config_contour, contour, contour_3D_gp

# configure contour of the mean
cfg = config_contour(y_train, y_mean, levels=20)
contour_and_scatter(X_train, y_train, X_test=X_test, y_test=y_mean.detach(),␣
 ↪label="Mean", config=cfg)

# configure contour of the uncertainty
cfg = config_contour(y_std.detach(), y_std.detach(), levels=20)
contour(X_test=X_test, y_test=y_std.detach(), label="Uncertainty", config=cfg)
contour_3D_gp(X_train=X_train, y_train=y_train,
              X_test=X_test, y_mean=y_mean.detach(), y_std=y_std.detach(),␣
 ↪levels=15)
```


Train data
x1
x2
Mean
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2.7138
2.4826
2.2515
2.0203
1.7892
1.5580
1.3269
1.0957
0.8646
0.6334
0.4023
0.1712
-0.0600
-0.2911
-0.5223
-0.7534
-0.9846
-1.2157
-1.4469
-1.6780


x2
x1
Uncertainty
0.1513
0.1448
0.1383
0.1318
0.1252
0.1187
0.1122
0.1056
0.0991
0.0926
0.0861
0.0795
0.0730
0.0665
0.0600
0.0534
0.0469
0.0404
0.0339
0.0273


Mean
2.7138
2.4001
2.0864
1.7727
1.4590
1.1453
0.8316
0.5179
0.2042
-0.1095
-0.4232
-0.7369
-1.0506
-1.3643
-1.6780