homework-08

August 8, 2025

1 Homework 8

1.1 References

- Lectures 27-28 (inclusive).

1.2 Instructions

- Type your name and email in the "Student details" section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

[ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
import seaborn as sns
sns.set_context("paper")
sns.set_style("ticks")

import scipy
import numpy as np
import scipy.stats as st
import urllib.request
import os

def download(
    url : str,
    local_filename : str = None
):
    """Download a file from a url.

    Arguments
    url            -- The url we want to download.
    local_filename -- The filemame to write on. If not
                      specified
```

```
    """
    if local_filename is None:
        local_filename = os.path.basename(url)
    urllib.request.urlretrieve(url, local_filename)
```

[ ]:
```
# Run this on Google colab
!pip install pyro-ppl
```

```
Collecting pyro-ppl
  Downloading pyro_ppl-1.9.1-py3-none-any.whl.metadata (7.8 kB)
Requirement already satisfied: numpy>=1.7 in /usr/local/lib/python3.11/dist-
packages (from pyro-ppl) (2.0.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.11/dist-packages (from pyro-ppl) (3.4.0)
Collecting pyro-api>=0.1.1 (from pyro-ppl)
  Downloading pyro_api-0.1.2-py3-none-any.whl.metadata (2.5 kB)
Requirement already satisfied: torch>=2.0 in /usr/local/lib/python3.11/dist-
packages (from pyro-ppl) (2.6.0+cu124)
Requirement already satisfied: tqdm>=4.36 in /usr/local/lib/python3.11/dist-
packages (from pyro-ppl) (4.67.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-
packages (from torch>=2.0->pyro-ppl) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0->pyro-ppl) (4.14.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-
packages (from torch>=2.0->pyro-ppl) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages
(from torch>=2.0->pyro-ppl) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages
(from torch>=2.0->pyro-ppl) (2025.3.2)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-
```

```
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0->pyro-ppl) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0->pyro-ppl) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch>=2.0->pyro-ppl) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch>=2.0->pyro-ppl)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-
packages (from torch>=2.0->pyro-ppl) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-
packages (from torch>=2.0->pyro-ppl) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=2.0->pyro-
ppl) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch>=2.0->pyro-ppl)
(3.0.2)
Downloading pyro_ppl-1.9.1-py3-none-any.whl (755 kB)
                         756.0/756.0 kB
32.6 MB/s eta 0:00:00
Downloading pyro_api-0.1.2-py3-none-any.whl (11 kB)
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4
MB)
                         363.4/363.4 MB
4.4 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (13.8 MB)
                         13.8/13.8 MB
97.0 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (24.6 MB)
                         24.6/24.6 MB
72.0 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (883 kB)
                         883.7/883.7 kB
```

```
48.3 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl
(664.8 MB)
                           664.8/664.8 MB
1.0 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl
(211.5 MB)
                           211.5/211.5 MB
5.8 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-
manylinux2014_x86_64.whl (56.3 MB)
                           56.3/56.3 MB
11.9 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-
manylinux2014_x86_64.whl (127.9 MB)
                           127.9/127.9 MB
7.6 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-
manylinux2014_x86_64.whl (207.5 MB)
                           207.5/207.5 MB
6.1 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (21.1 MB)
                           21.1/21.1 MB
87.2 MB/s eta 0:00:00
Installing collected packages: pyro-api, nvidia-nvjitlink-cu12, nvidia-
curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-
cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-cusparse-cu12, nvidia-
cudnn-cu12, nvidia-cusolver-cu12, pyro-ppl
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
```

```
    Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-
cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127
nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-
cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170
nvidia-nvjitlink-cu12-12.4.127 pyro-api-0.1.2 pyro-ppl-1.9.1
```

[ ]:
```python
import pyro
import pyro.distributions as dist
from pyro.infer import MCMC, NUTS
import torch
```

1.3 Student details

- **First Name:** Hannah
- **Last Name:** Moskios
- **Email:** hmoskios@purdue.edu

1.4 Problem 1 - Bayesian Linear regression on steroids

The purpose of this problem is to demonstrate that we have learned enough to do very complicated things. In the first part, we will do Bayesian linear regression with radial basis functions (RBFs) in which we characterize the posterior of all parameters, including the length-scales of the RBFs. In the second part, we are going to build a model that has an input-varying noise. Such models are called heteroscedastic models.

We need to write some `pytorch` code to compute the design matrix. This is absolutely necessary so that `pyro` can differentiate through all expressions.

[ ]:
```python
class RadialBasisFunctions(torch.nn.Module):
    """Radial basis functions basis.

    Arguments:
    X   -  The centers of the radial basis functions.
    ell -  The assumed length scale.
    """
    def __init__(self, X, ell):
        super().__init__()
        self.X = X
        self.ell = ell
        self.num_basis = X.shape[0]
    def forward(self, x):
        distances = torch.cdist(x, self.X)
        return torch.exp(-.5 * distances ** 2 / self.ell ** 2)
```

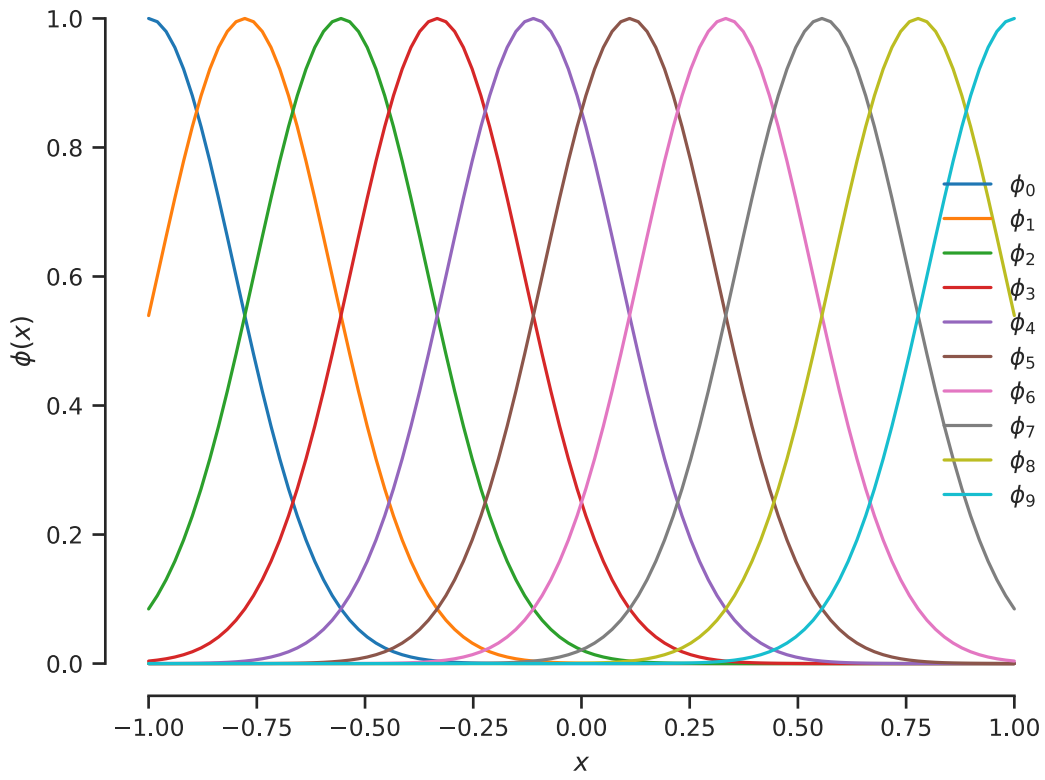
Here is how you can use them:

[ ]:
```python
# Make the basis
x_centers = torch.linspace(-1, 1, 10).unsqueeze(-1)
ell = 0.2
basis = RadialBasisFunctions(x_centers, ell)

# Some points (need to be N x 1)
x = torch.linspace(-1, 1, 100).unsqueeze(-1)

# Evaluate the basis
Phi = basis(x)

# Here is the shape of Phi
print(Phi.shape)
```

```
torch.Size([100, 10])
```

Here is how they look like:

[ ]:
```python
fig, ax = plt.subplots()
for i in range(Phi.shape[1]):
    ax.plot(x, Phi[:, i], label=f"$\phi_{i}$")
ax.set(xlabel="$x$", ylabel="$\phi(x)$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```



1.4.1 Part A - Hierarchical Bayesian linear regression with input-independent noise

We will analyze the motorcycle dataset. The data is loaded below.

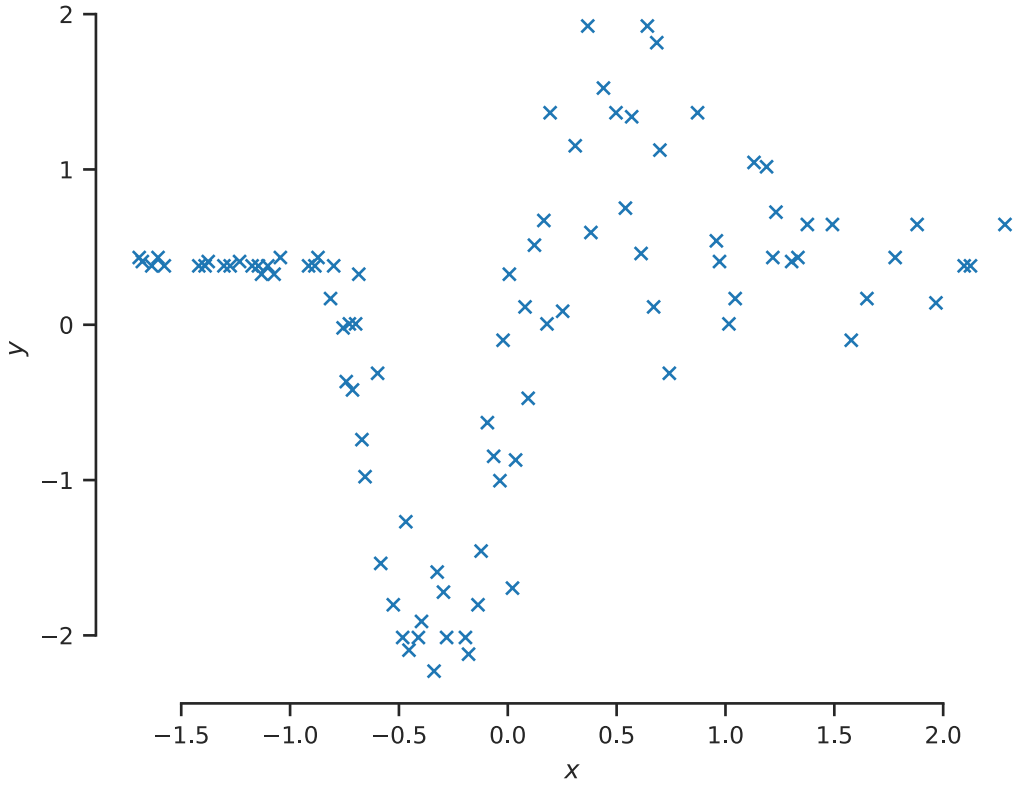
[ ]:
```
url = "https://raw.githubusercontent.com/PredictiveScienceLab/data-analytics-se/
↪master/lecturebook/data/motor.dat"
!curl -O $url
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100  2970  100  2970    0     0  22093      0 --:--:-- --:--:-- --:--:-- 22164
```

We will work with the scaled data:

[ ]:
```
from sklearn.preprocessing import StandardScaler

data = np.loadtxt('motor.dat')
scaler = StandardScaler()
data = scaler.fit_transform(data)
X = torch.tensor(data[:, 0], dtype=torch.float32).unsqueeze(-1)
Y = torch.tensor(data[:, 1], dtype=torch.float32)
```

```
fig, ax = plt.subplots()
ax.plot(X, Y, 'x')
ax.set(xlabel="$x$", ylabel="$y$")
sns.despine(trim=True);
```



1.4.2 Part A.I

Your goal is to implement the model described below. We use the radial basis functions (`RadialBasisFunction`) with centers, x_i at $m = 50$ equidistant points between the minimum and maximum of the observed inputs:

$$\phi_i(x; \ell) = \exp\left(-\frac{(x - x_i)^2}{2\ell^2}\right),$$

for $i = 1, \ldots, m$. We denote the vector of RBFs evaluated at x as $\phi(x; \ell)$.

We are not going to pick the length-scales ℓ by hand. Instead, we will put a prior on it:

$$\ell \sim \text{Exponential}(1).$$

The corresponding weights have priors:

$$w_j|\alpha_i \sim N(0, \alpha_j^2),$$

and its α_j has a prior:

$$\alpha_j \sim \text{Exponential}(1),$$

for $j = 1, \ldots, m$.

Denote our data as:

$$x_{1:n} = (x_1, \ldots, x_n)^T, \text{ (inputs)},$$

and

$$y_{1:n} = (y_1, \ldots, y_n)^T, \text{ (outputs)}.$$

The likelihood of the data is:

$$y_i|\mathbf{w}, \sigma \sim N(\mathbf{w}^T \phi(x_i; \ell), \sigma^2),$$

for $i = 1, \ldots, n$.

$$y_n|\ell, \mathbf{w}, \sigma \sim N(\mathbf{w}^T \phi(x_n; \ell), \sigma^2).$$

Complete the `pyro` implementation of that model:

Answer:

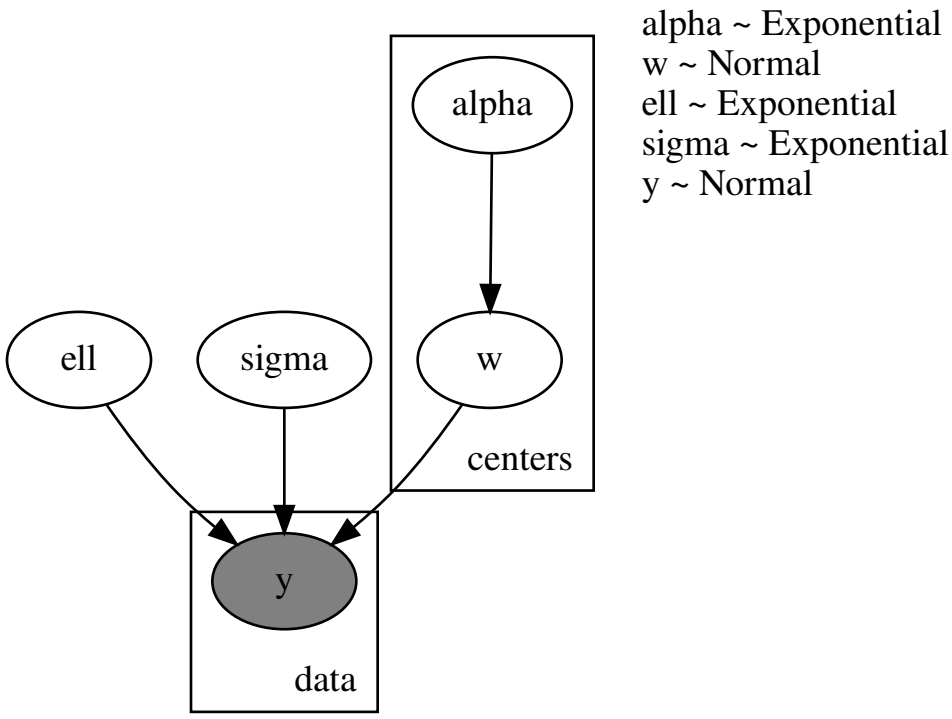
[ ]:
```
def model(X, y, num_centers=50):
    with pyro.plate("centers", num_centers):
        alpha = pyro.sample("alpha", dist.Exponential(1.0))
        # Notice below that dist.Normal needs the standard deviation - not the␣
↪variance
        # We follow a different convention in the lecture notes
        w = pyro.sample("w", dist.Normal(0.0, alpha))
    # Complete the code assign to ell the correct prior distribution (an␣
↪Exponential(1)).
    ell = pyro.sample("ell", dist.Exponential(1.0))
    # Hint: Look at alpha.
    # Complete the code assign to sigma the correct prior distribution (an␣
↪Exponential(1)).
    sigma = pyro.sample("sigma", dist.Exponential(1.0))
    x_centers = torch.linspace(X.min(), X.max(), num_centers).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(X)
```

```
    with pyro.plate("data", X.shape[0]):
        pyro.sample("y", dist.Normal(Phi @ w, sigma), obs=y)
    # Notice that I'm making the model return all the variables that I have␣
↪made.
    # This is not essential for characterizing the posterior, but it does␣
↪reduce redundant code
    # when we are trying to get the posterior predictive.
    return locals()
```

The graph will help to understand the model:

[ ]:
```
pyro.render_model(model, (X, Y), render_distributions=True)
```

[ ]:



Use `pyro.infer.autoguide.AutoDiagonalNormal` to make the guide:

[ ]:
```
guide = pyro.infer.autoguide.AutoDiagonalNormal(model)
```

We will use variational inference. Here is the training code from the hans-on activity:

[ ]:
```
def train(model, guide, data, num_iter=5_000):
    """Train a model with a guide.

    Arguments
    ---------
```

```
    model    -- The model to train.
    guide    -- The guide to train.
    data     -- The data to train the model with.
    num_iter -- The number of iterations to train.

    Returns
    -------
    elbos -- The ELBOs for each iteration.
    param_store -- The parameters of the model.
    """

    pyro.clear_param_store()

    optimizer = pyro.optim.Adam({"lr": 0.001})

    svi = pyro.infer.SVI(
        model,
        guide,
        optimizer,
        loss=pyro.infer.JitTrace_ELBO()
    )

    elbos = []
    for i in range(num_iter):
        loss = svi.step(*data)
        elbos.append(-loss)
        if i % 1_000 == 0:
            print(f"Iteration: {i} Loss: {loss}")

    return elbos, pyro.get_param_store()
```

1.4.3 Part A.II

Train the model for 20,000 iterations. Call the `train()` function we defined above to do it. Make sure you store the returned elbo values because you will need them later.

Answer:

[ ]:
```
# Your code here
num_iter = 20000
elbos, params = train(model, guide, (X, Y), num_iter=num_iter)
```

```
Iteration: 0 Loss: 417.47564697265625
Iteration: 1000 Loss: 212.980224609375
Iteration: 2000 Loss: 162.26339721679688
Iteration: 3000 Loss: 134.44354248046875
Iteration: 4000 Loss: 119.58905792236328
Iteration: 5000 Loss: 134.87353515625
```

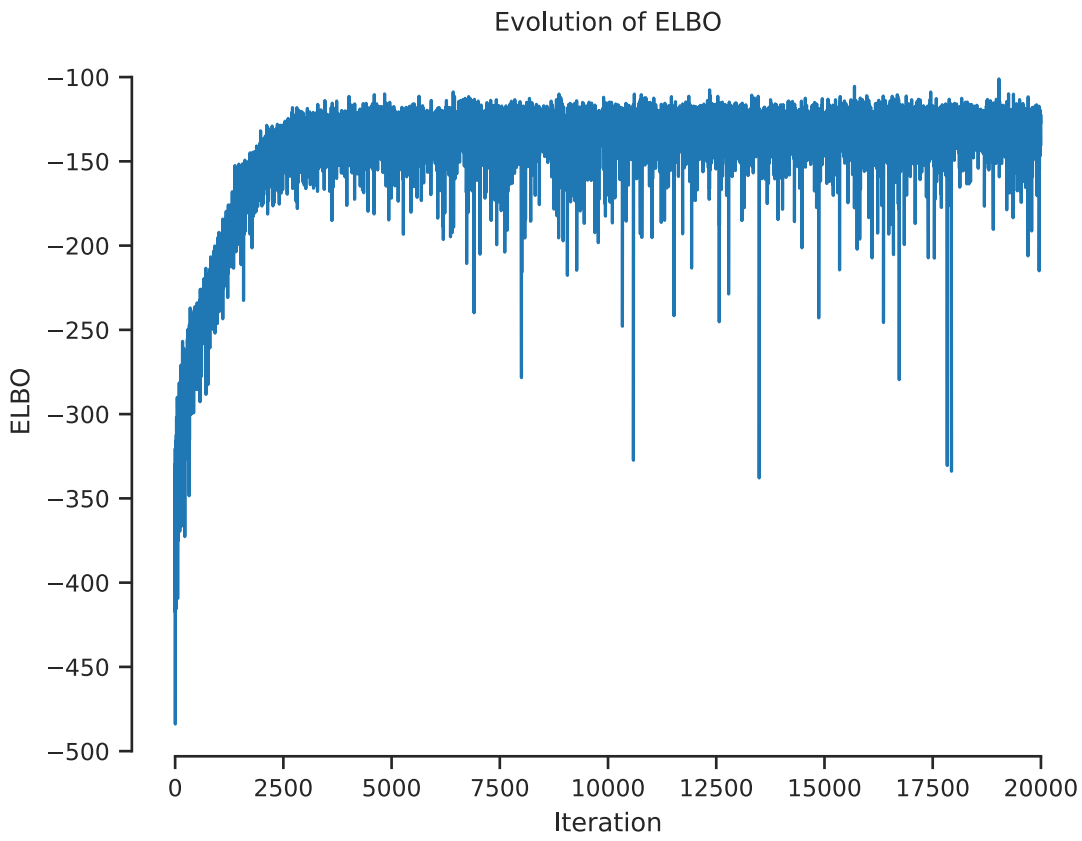
```
Iteration: 6000 Loss: 130.06527709960938
Iteration: 7000 Loss: 114.22335815429688
Iteration: 8000 Loss: 278.37481689453125
Iteration: 9000 Loss: 133.55001831054688
Iteration: 10000 Loss: 133.69186401367188
Iteration: 11000 Loss: 122.07220458984375
Iteration: 12000 Loss: 130.49038696289062
Iteration: 13000 Loss: 126.03202819824219
Iteration: 14000 Loss: 137.73220825195312
Iteration: 15000 Loss: 131.58302307128906
Iteration: 16000 Loss: 133.30642700195312
Iteration: 17000 Loss: 118.59111785888672
Iteration: 18000 Loss: 125.1243896484375
Iteration: 19000 Loss: 116.53839874267578
```

1.4.4 Part A.III

Plot the evolution of the ELBO.

Answer:

[ ]:
```python
# Your code here
fig, ax = plt.subplots()
ax.plot(elbos)
ax.set(xlabel="Iteration", ylabel="ELBO", title="Evolution of ELBO")
sns.despine(trim=True);
```



1.4.5 Part A.IV

Take 1,000 posterior samples.

Answer:

I'm giving you this one because it is a bit tricky. You need to use the `pyro.infer.Predictive` class to do it. Here is how you can use it:

[ ]:
```
post_samples = pyro.infer.Predictive(model, guide=guide, num_samples=1000)(X, Y)
# Just modify the call to get the right number of samples
```

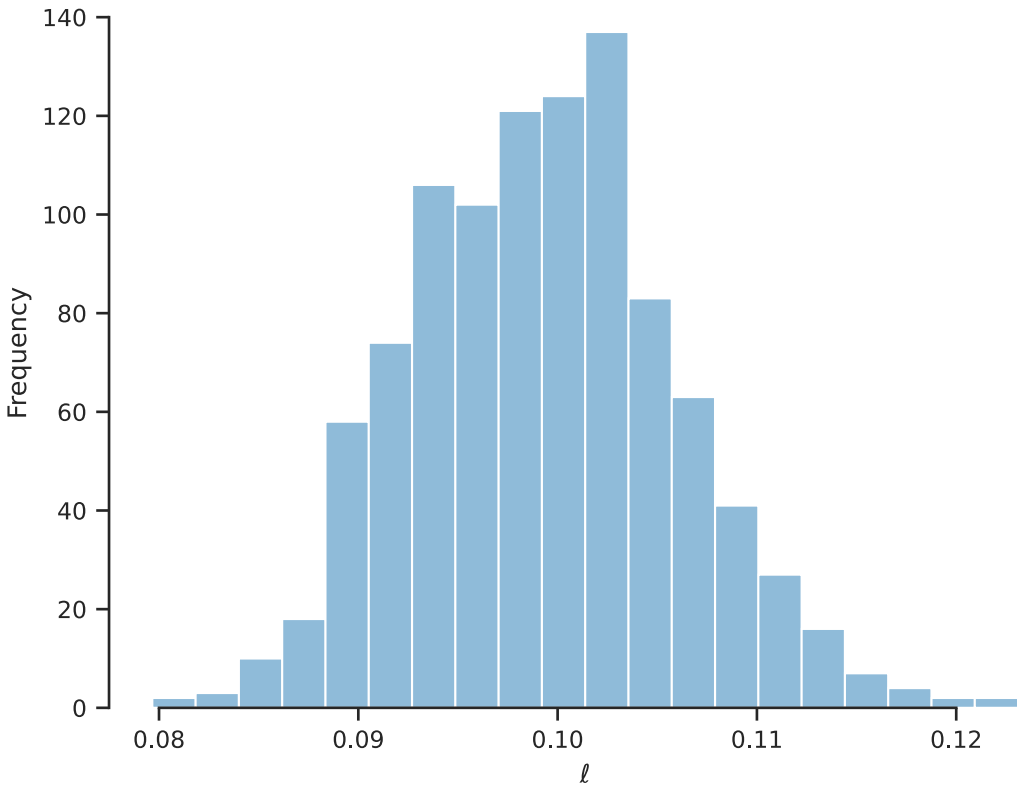
1.4.6 Part A.V

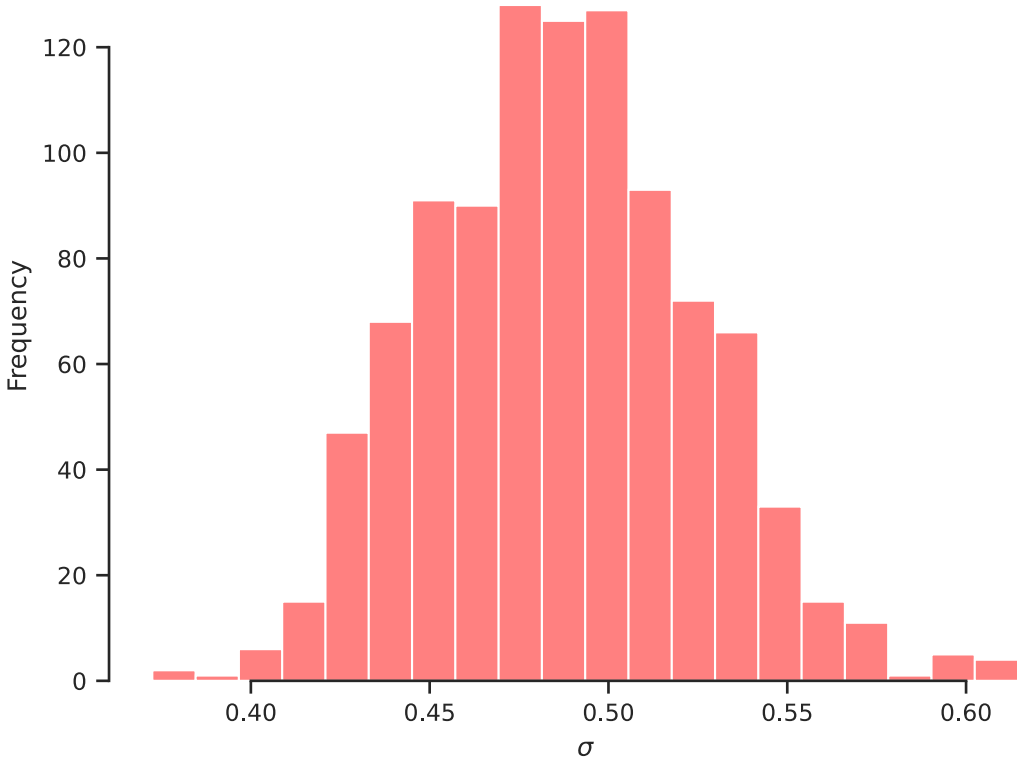
Plot the histograms of the posteriors of ℓ, σ, α_{10} and w_{10}.

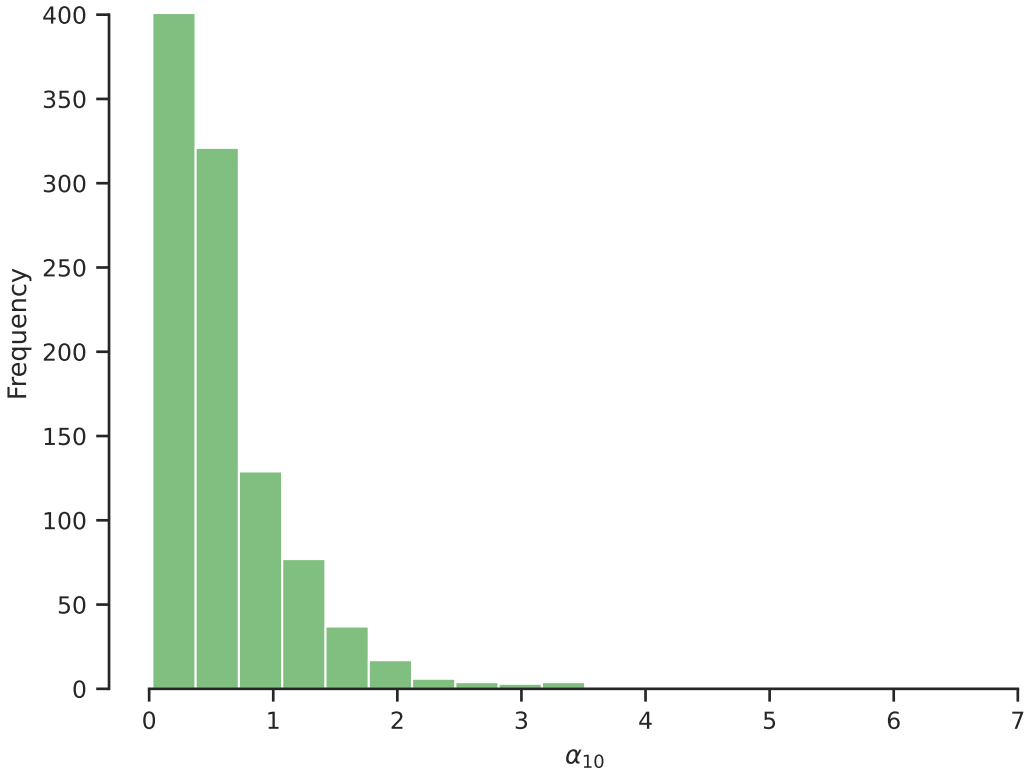
Answer:

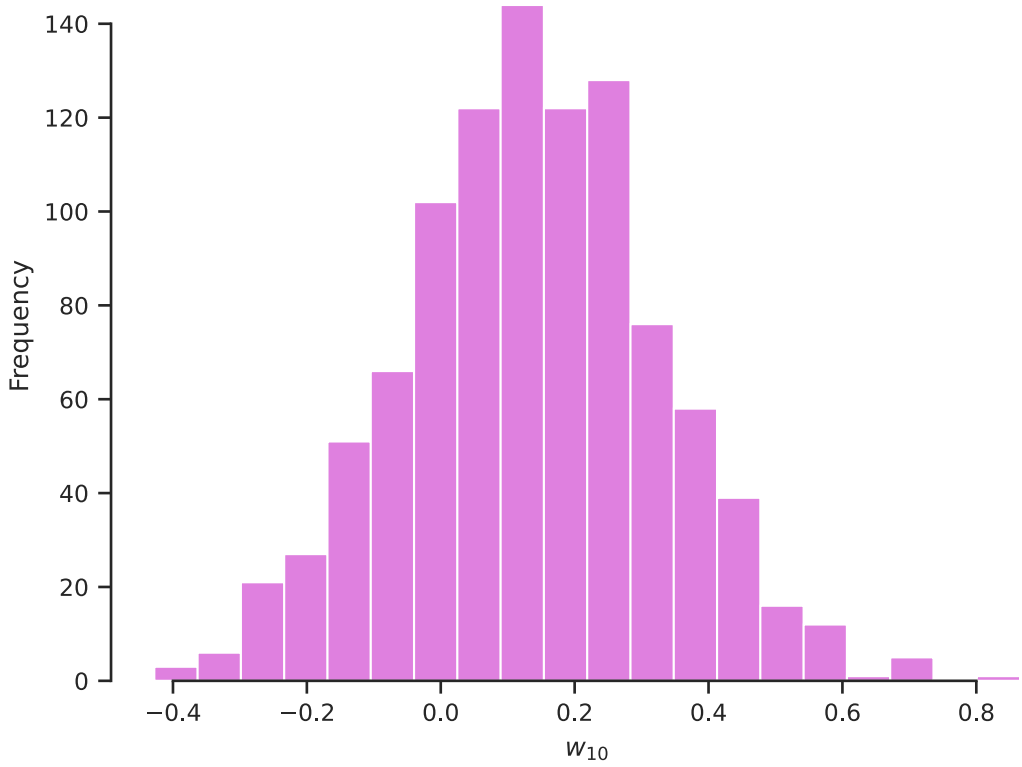
[ ]:
```
# First, here is how to extract the samples.
ell = post_samples["ell"]
# You can do `post_samples.keys()` to see all the keys.
# But they should correspond to the names of the latent variables in the model.
```

```
sigma = post_samples["sigma"]   # Your code here
alphas = post_samples["alpha"]  # Your code here
ws = post_samples["w"]          # Your code here

# Here is the code to make the histogram for the length scale.
fig, ax = plt.subplots()
# **VERY IMPORTANT** - You need to detach the tensor from the computational␣
 ↪graph.
# Otherwise, you will get very very strange behavior.
ax.hist(ell.detach().numpy(), bins=20, alpha=.5)
ax.set(xlabel="$\ell$", ylabel="Frequency")
sns.despine(trim=True);

# Your code for the other histograms here
# Histogram for sigma
fig, ax = plt.subplots()
ax.hist(sigma.detach().numpy(), bins=20, alpha=.5, color='r')
ax.set(xlabel="$\sigma$", ylabel="Frequency")
sns.despine(trim=True);

# Histogram for alpha_10
fig, ax = plt.subplots()
ax.hist(alphas[:, 9].detach().numpy(), bins=20, alpha=.5, color='g')
ax.set(xlabel=r"$\alpha_{10}$", ylabel="Frequency")
sns.despine(trim=True);

# Histogram for w_10
fig, ax = plt.subplots()
ax.hist(ws[:, 9].detach().numpy(), bins=20, alpha=.5, color='m')
ax.set(xlabel="$w_{10}$", ylabel="Frequency")
sns.despine(trim=True);
```


140
120
100
80
60
40
20
0
Frequency
0.08
0.09
0.10
0.11
0.12
ℓ


120
100
80
60
40
20
0
Frequency
0.40
0.45
0.50
0.55
0.60
σ


400
350
300
250
200
150
100
50
0
Frequency
0
1
2
3
4
5
6
7
α_{10}



1.4.7 Part A.VI

Let's extend them model to make predictions.

Answer:

[ ]:
```
# Again, I'm giving you most of the code here.

def predictive_model(X, y, num_centers=50):
    # First we run the original model get all the variables
    params = model(X, y, num_centers)
    # Here is how you can access the variables
    w = params["w"]
    ell = params["ell"]  # Access the length scale
    sigma = params["sigma"]  # Access the standard deviation of the noise
    x_centers = params["x_centers"]  # Access centers of radial basis functions
    # Here are the points where we want to make predictions
    xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
    # Evaluate the basis on the prediction points
    Phi = RadialBasisFunctions(x_centers, ell)(xs)
    # Make the predictions - we use a deterministic node here because we want
    # to save the results of the predictions.
```

```
    predictions = pyro.deterministic("predictions", Phi @ w)
    # Finally, we add the measurement noise
    predictions_with_noise = pyro.sample("predictions_with_noise", dist.
↪Normal(predictions, sigma))
    return locals()
```

1.4.8 Part A.VII

Extract the posterior predictive distribution using 10,000 samples. Separate aleatory and epistemic uncertainty.

Answer:

[ ]:
```
# Here is how to make the predictions. Just change the number of samples to the␣
↪right number.
post_pred = pyro.infer.Predictive(predictive_model, guide=guide,␣
↪num_samples=10000)(X, Y)
# We will predict here:
xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
# You can extract the predictions from post_pred like this:
predictions = post_pred["predictions"]
# Note that we extracted the deterministic node called "predictions" from the␣
↪model.
# Get the epistemic uncertainty in the usual way:
p_500, p_025, p_975 = np.percentile(predictions, [50, 2.5, 97.5], axis=0)
# Extract predictions with noise (your code here)
predictions_with_noise = post_pred["predictions_with_noise"]
# Get the aleatory uncertainty (your code here)
ap_025, ap_975 = np.percentile(predictions_with_noise, [2.5, 97.5], axis=0)
```

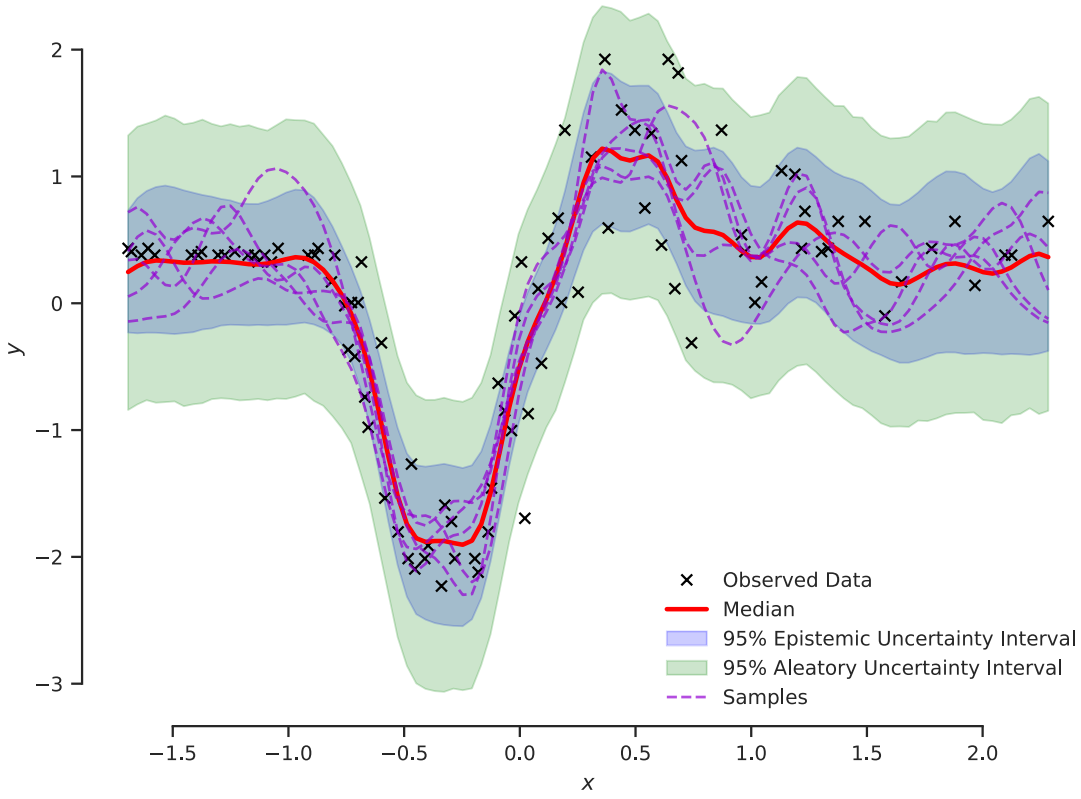
1.4.9 Part A.VIII

Plot the data, the median, the 95% credible interval of epistemic uncertainty and the 95% credible interval of aleatory uncertainty, along with five samples from the posterior.

Answer:

[ ]:
```
# Your code here. You have everything you need to make the plot.
# Plot the observed data
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(X, Y, 'kx', label="Observed Data")

# Plot the median
ax.plot(xs.flatten(), p_500.flatten(), color="r", lw=2, label="Median")

# Plot the epistemic uncertainty interval
ax.fill_between(xs.flatten(), p_025.flatten(), p_975.flatten(), color="b",
                alpha=.2, label="95% Epistemic Uncertainty Interval")
```

```
# Plot the aleatory uncertainty interval
ax.fill_between(xs.flatten(), ap_025.flatten(), ap_975.flatten(), color="g",
                alpha=.2, label="95% Aleatory Uncertainty Interval")

# Plot 5 samples from the posterior
for i in range(5):
    sample_idx = np.random.choice(predictions.shape[0])
    sample = predictions[sample_idx]
    if i == 0:
        ax.plot(xs.flatten(), sample.flatten(), '--',
                color="darkviolet", alpha=.7, label="Samples")
    else:
        ax.plot(xs.flatten(), sample.flatten(), '--',
                color="darkviolet", alpha=.7)

# Add axis labels and legend
ax.set(xlabel="$x$", ylabel="$y$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```



1.4.10 Part B - Heteroscedastic regression

We are going to build a model that has an input-varying noise. Such models are called heteroscedastic models. Here I will let you do more of the work.

Everything is as before for ℓ, the α_j's, and the w_j's. We now introduce a model for the noise that is input dependent. It will use the same RBFs as the mean function. But let's use a different length-scale, ℓ_σ. So, we add:

$$\ell_\sigma \sim \text{Exponential}(1),$$

$$\alpha_{\sigma,j} \sim \text{Exponential}(1),$$

and

$$w_{\sigma,j}|\alpha_{\sigma,j} \sim N(0, \alpha_{\sigma,j}^2),$$

for $j = 1, \ldots, m$.

Our model for the input-dependent noise variance is:

$$\sigma(x; \mathbf{w}_\sigma, \ell) = \exp\left(\mathbf{w}_\sigma^T \phi(x; \ell_\sigma)\right).$$

So, the likelihood of the data is:

$$y_i|\mathbf{w}, \mathbf{w}_\sigma \sim N\left(\mathbf{w}^T \phi(x_i; \ell), \sigma^2(x_i; \mathbf{w}_\sigma, \ell)\right),$$

You will implement this model.

1.4.11 Part B.I

Complete the code below:

[ ]:
```
def model(X, y, num_centers=50):
    with pyro.plate("centers", num_centers):
        alpha = pyro.sample("alpha", dist.Exponential(1.0))
        w = pyro.sample("w", dist.Normal(0.0, alpha))
        # Let's add the generalized linear model for the log noise (your code).
        alpha_noise = pyro.sample("alpha_noise", dist.Exponential(1.0))
        w_noise = pyro.sample("w_noise", dist.Normal(0.0, alpha_noise))
    ell = pyro.sample("ell", dist.Exponential(1.))
    ell_noise = pyro.sample("ell_noise", dist.Exponential(1.0)) # Your code here
    x_centers = torch.linspace(X.min(), X.max(), num_centers).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(X)
    Phi_noise = RadialBasisFunctions(x_centers, ell_noise)(X) # Your code here
    # This is the new part 2/2
    model_mean = Phi @ w
```

```
    sigma = torch.exp(Phi_noise @ w_noise)  # Your code here (torch.exp())
    with pyro.plate("data", X.shape[0]):
        pyro.sample("y", dist.Normal(model_mean, sigma), obs=y)
    return locals()
```

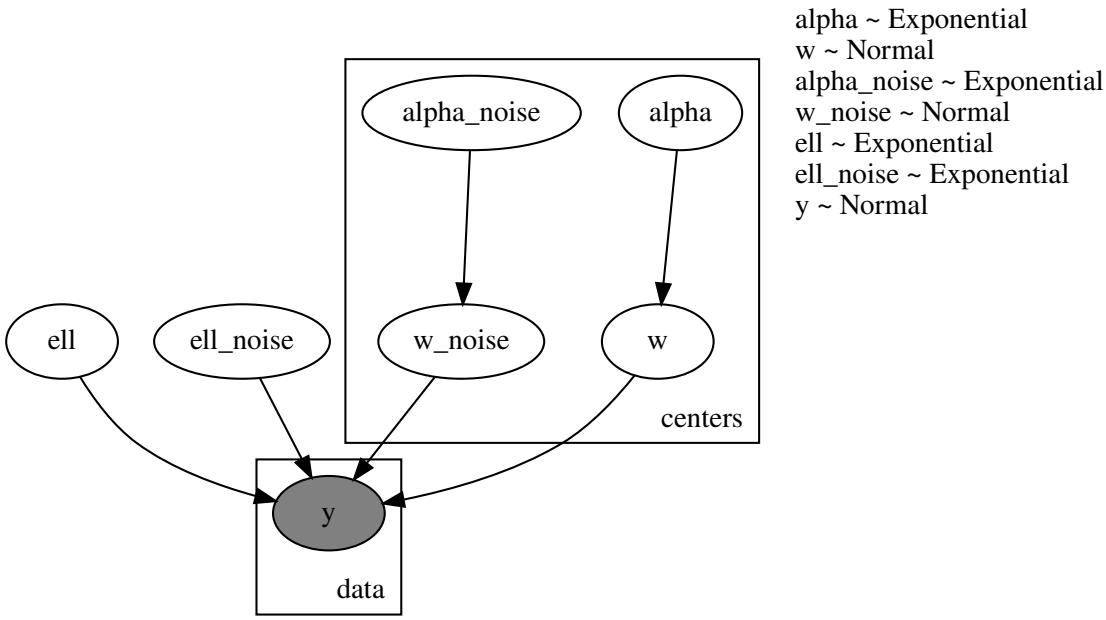
Make a `pyro.infer.autoguide.AutoDiagonalNormal` guide:

[ ]:
```
# Your code here
guide = pyro.infer.autoguide.AutoDiagonalNormal(model)
```

Make the graph of the model using `pyro` functionality:

[ ]:
```
# Your code here
pyro.render_model(model, (X, Y), render_distributions=True)
```

[ ]:



1.4.12 Part B.II

Train the model using 20,000 iterations. Then plot the evolution of the ELBO.

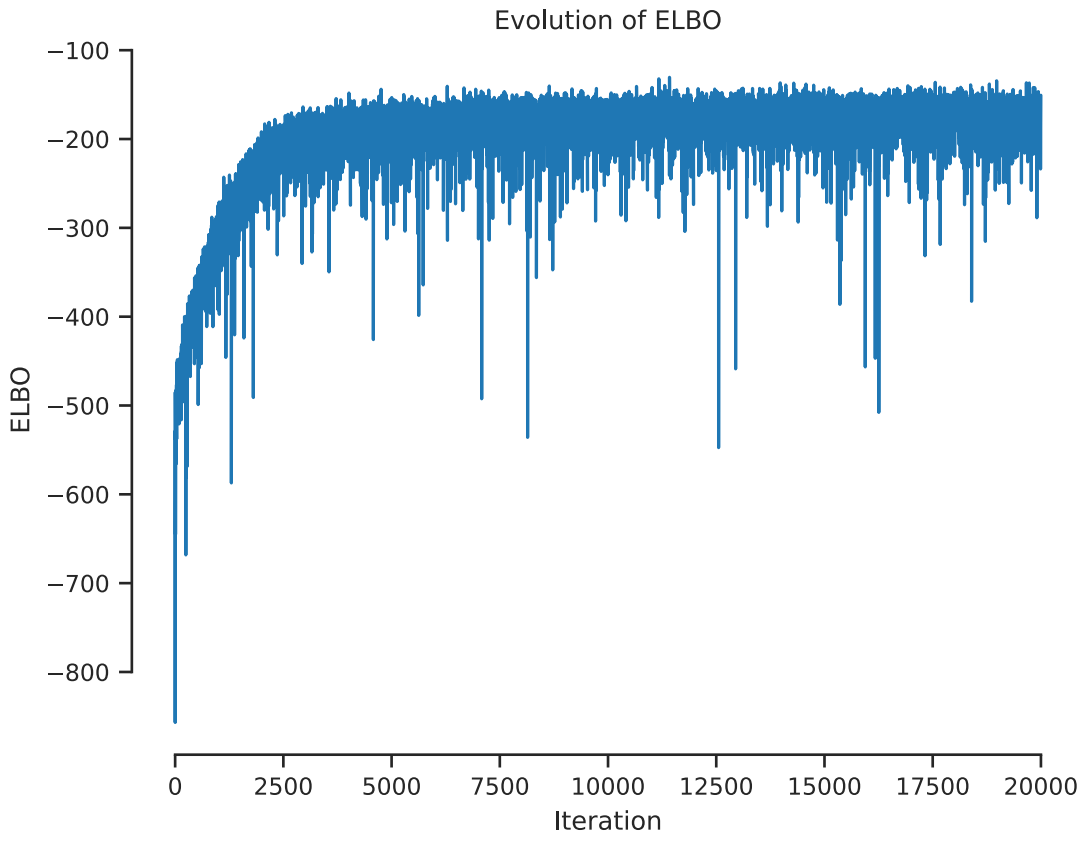
Answer:

[ ]:
```
# Your code here
num_iter = 20000
elbos, params = train(model, guide, (X, Y), num_iter=num_iter)
```

```
Iteration: 0 Loss: 529.8273315429688
Iteration: 1000 Loss: 307.93157958984375
Iteration: 2000 Loss: 229.0401153564453
```

```
Iteration: 3000 Loss: 191.35794067382812
Iteration: 4000 Loss: 178.6262969970703
Iteration: 5000 Loss: 187.37762451171875
Iteration: 6000 Loss: 182.12391662597656
Iteration: 7000 Loss: 176.38307189941406
Iteration: 8000 Loss: 206.5997314453125
Iteration: 9000 Loss: 168.88316345214844
Iteration: 10000 Loss: 181.05615234375
Iteration: 11000 Loss: 183.99307250976562
Iteration: 12000 Loss: 171.5192108154297
Iteration: 13000 Loss: 150.842041015625
Iteration: 14000 Loss: 160.65243530273438
Iteration: 15000 Loss: 173.86251831054688
Iteration: 16000 Loss: 175.39874267578125
Iteration: 17000 Loss: 156.70208740234375
Iteration: 18000 Loss: 162.9134521484375
Iteration: 19000 Loss: 178.73423767089844
```

[ ]:
```
# Plot the evolution of the ELBO
fig, ax = plt.subplots()
ax.plot(elbos)
ax.set(xlabel="Iteration", ylabel="ELBO", title="Evolution of ELBO")
sns.despine(trim=True);
```



1.4.13 Part B.III

Extend the model to make predictions.

Answer:

[ ]:
```
def predictive_model(X, y, num_centers=50):
    params = model(X, y, num_centers)
    w = params["w"]
    w_noise = params["w_noise"]      # Your code here
    ell = params["ell"]              # Your code here
    ell_noise = params["ell_noise"]  # Your code here
    sigma = params["sigma"]          # Your code here
    x_centers = params["x_centers"]
    xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(xs)             # Your code here
    Phi_noise = RadialBasisFunctions(x_centers, ell_noise)(xs) # Your code here
    predictions = pyro.deterministic("predictions", Phi @ w)
    # Your code here (pyro.deterministic("sigma", <something>))
    sigma = pyro.deterministic("sigma", torch.exp(Phi_noise @ w_noise))
    predictions_with_noise = pyro.sample("predictions_with_noise", dist.
 ↪Normal(predictions, sigma))  # Your code here
    return locals()
```

1.4.14 Part B.IV

Now, make predictions and calculate the epistemic and aleatory uncertainties as in part A.VII.

Answer:

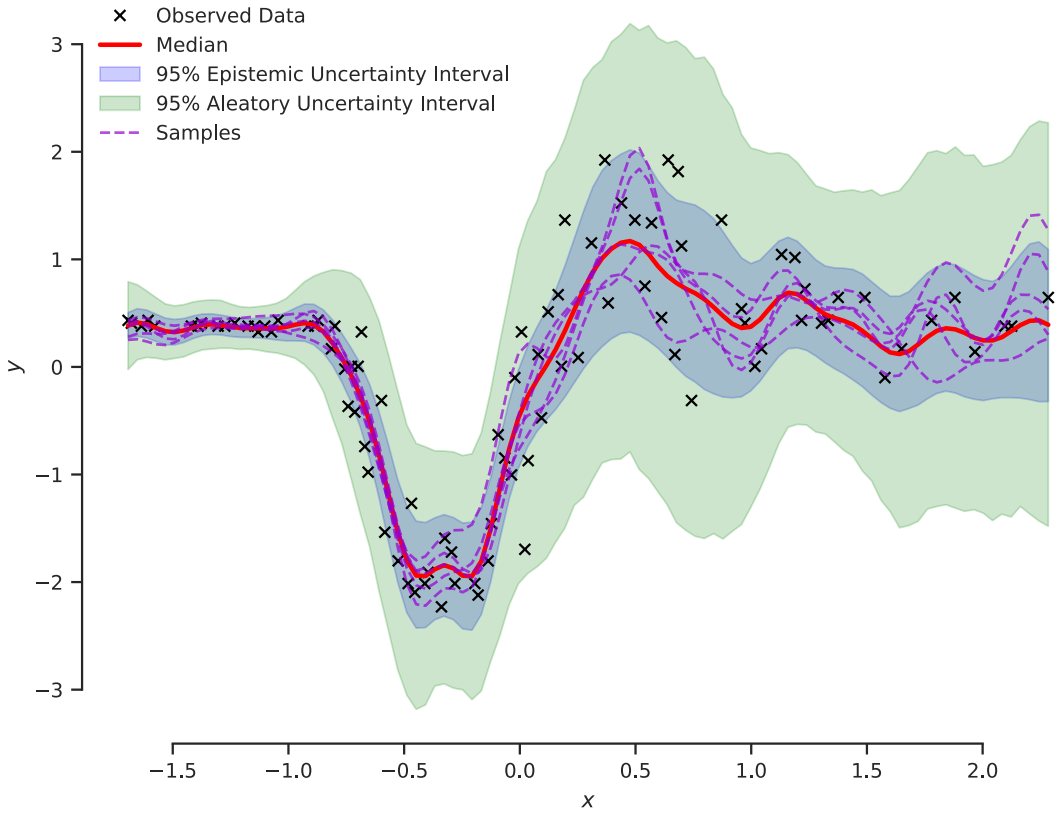
[ ]:
```
# Your code here
post_pred = pyro.infer.Predictive(predictive_model, guide=guide,␣
 ↪num_samples=10000)(X, Y)
# We will predict here:
xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
# Extract the predictions and get the epistemic uncertainty
predictions = post_pred["predictions"]
p_500, p_025, p_975 = np.percentile(predictions, [50, 2.5, 97.5], axis=0)
# Extract predictions with noise and get the aleatory uncertainty
predictions_with_noise = post_pred["predictions_with_noise"]
ap_025, ap_975 = np.percentile(predictions_with_noise, [2.5, 97.5], axis=0)
```

1.4.15 Part B.V

Make the same plot as in part A.VIII.

Answer:

[ ]:
```
# Your code here
# Plot the observed data
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(X, Y, 'kx', label="Observed Data")

# Plot the median
ax.plot(xs.flatten(), p_500.flatten(), color="r", lw=2, label="Median")

# Plot the epistemic uncertainty interval
ax.fill_between(xs.flatten(), p_025.flatten(), p_975.flatten(), color="b",
                alpha=.2, label="95% Epistemic Uncertainty Interval")

# Plot the aleatory uncertainty interval
ax.fill_between(xs.flatten(), ap_025.flatten(), ap_975.flatten(), color="g",
                alpha=.2, label="95% Aleatory Uncertainty Interval")

# Plot 5 samples from the posterior
for i in range(5):
    sample_idx = np.random.choice(predictions.shape[0])
    sample = predictions[sample_idx]
    if i == 0:
        ax.plot(xs.flatten(), sample.flatten(), '--',
                color="darkviolet", alpha=.7, label="Samples")
    else:
        ax.plot(xs.flatten(), sample.flatten(), '--',
                color="darkviolet", alpha=.7)

# Add axis labels and legend
ax.set(xlabel="$x$", ylabel="$y$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```

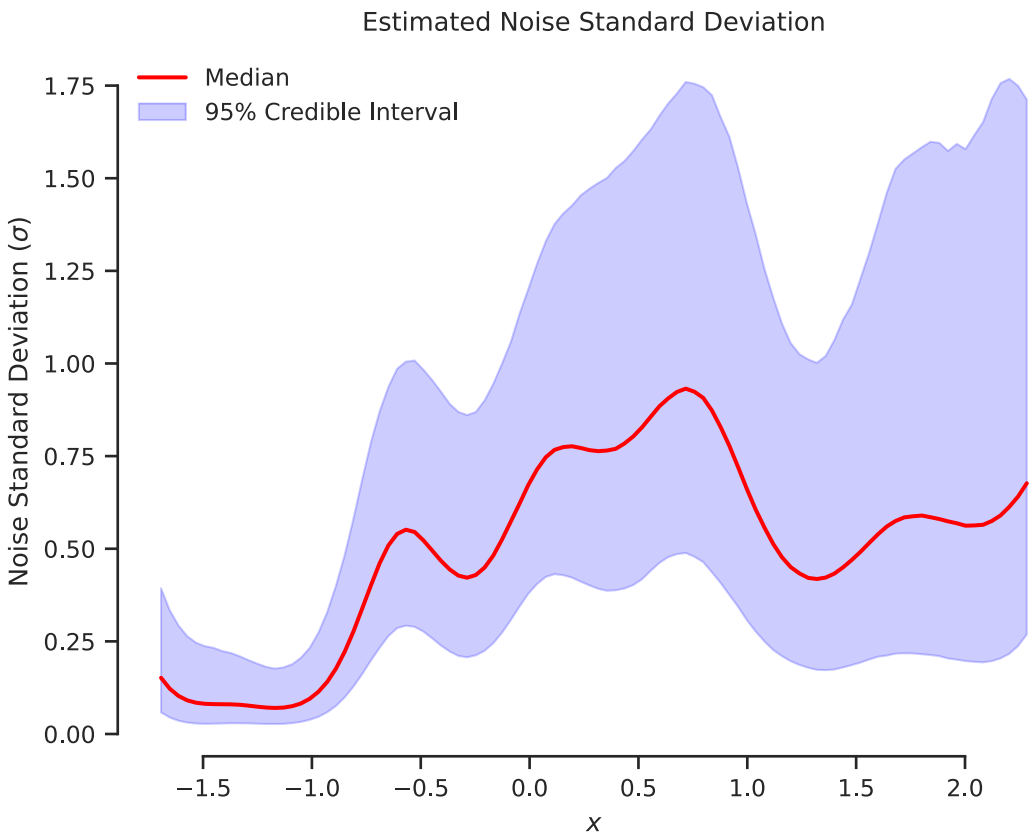


1.4.16 Part B.VI

Plot the estimated noise standard deviation as a function of of the input along with a 95% credible interval.

Answer:

[ ]:
```
# Your code here
# Compute the percentiles for the noise standard deviation
sigma_025, sigma_500, sigma_975 = np.percentile(post_pred["sigma"], [2.5, 50,␣
↪97.5], axis=0)

# Plot the estimated noise standard deviation and 95% credible interval
fig, ax = plt.subplots()
ax.plot(xs.flatten(), sigma_500.flatten(), color="r", lw=1.5, label="Median")
ax.fill_between(xs.flatten(), sigma_025.flatten(), sigma_975.flatten(),
                color="b", alpha=.2, label="95% Credible Interval")
ax.set(xlabel="$x$", ylabel="Noise Standard Deviation ($\sigma$)",
       title="Estimated Noise Standard Deviation")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```



1.4.17 Part B.VII

Which model do you prefer? Why?

Answer: I prefer the heteroscedastic model because it more accurately captures the noise in the underlying data. The 'motor.dat' dataset consists of 94 points divided into three strata, where each stratum has a different variance residual estimate. This means there are varying noise levels across the dataset. The model in part A is homoscedastic, meaning it assumes the entire dataset has constant noise. In contrast, the model is part B is heteroscedastic, meaning it has input-varying noise. The noise variations in the heteroscedastic model are evident in part B.VI. Therefore, the heteroscedastic model better fits the noise in the observed data.

1.4.18 Part B.IX

Can you think of any way to improve the model? Go crazy! This is the last homework assignment! There is no right or wrong answer here. But if you have a good idea, we will give you extra credit.

How I Will Improve the Model:

Instead of fixing RBF centers uniformly in input space, we can treat the centers as learnable parameters using `pyro.param`. This gives the optimizer full freedom to locate centers where they best support the data. Specifically, this allows the model to adaptively shift basis functions to-

ward regions that require more expressiveness. By letting the data inform where basis functions are placed, the model can make better predictions and avoid underfitting in areas with complex structure.

[ ]:
```
## Your code and answers here
def model(X, y, num_centers=50):
    with pyro.plate("centers", num_centers):
        alpha = pyro.sample("alpha", dist.Exponential(1.0))
        w = pyro.sample("w", dist.Normal(0.0, alpha))

        alpha_noise = pyro.sample("alpha_noise", dist.Exponential(1.0))
        w_noise = pyro.sample("w_noise", dist.Normal(0.0, alpha_noise))

    ell = pyro.sample("ell", dist.Exponential(1.))
    ell_noise = pyro.sample("ell_noise", dist.Exponential(1.0))

    # Make the centers learnable parameters using pyro.param
    x_centers = pyro.param("x_centers", torch.linspace(X.min(), X.max(),␣
 ↪num_centers).unsqueeze(-1))
    Phi = RadialBasisFunctions(x_centers, ell)(X)
    Phi_noise = RadialBasisFunctions(x_centers, ell_noise)(X)

    model_mean = Phi @ w
    sigma = torch.exp(Phi_noise @ w_noise)
    with pyro.plate("data", X.shape[0]):
        pyro.sample("y", dist.Normal(model_mean, sigma), obs=y)
    return locals()
```

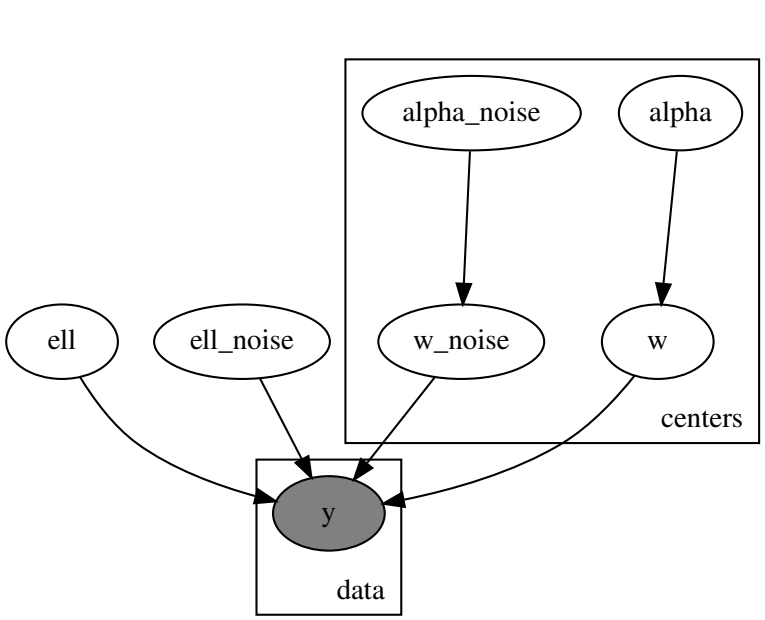
Make a `pyro.infer.autoguide.AutoDiagonalNormal` guide:

[ ]:
```
guide = pyro.infer.autoguide.AutoDiagonalNormal(model)
```

Make the graph of the model using `pyro` functionality:

[ ]:
```
pyro.render_model(model, (X, Y), render_distributions=True)
```

[ ]:



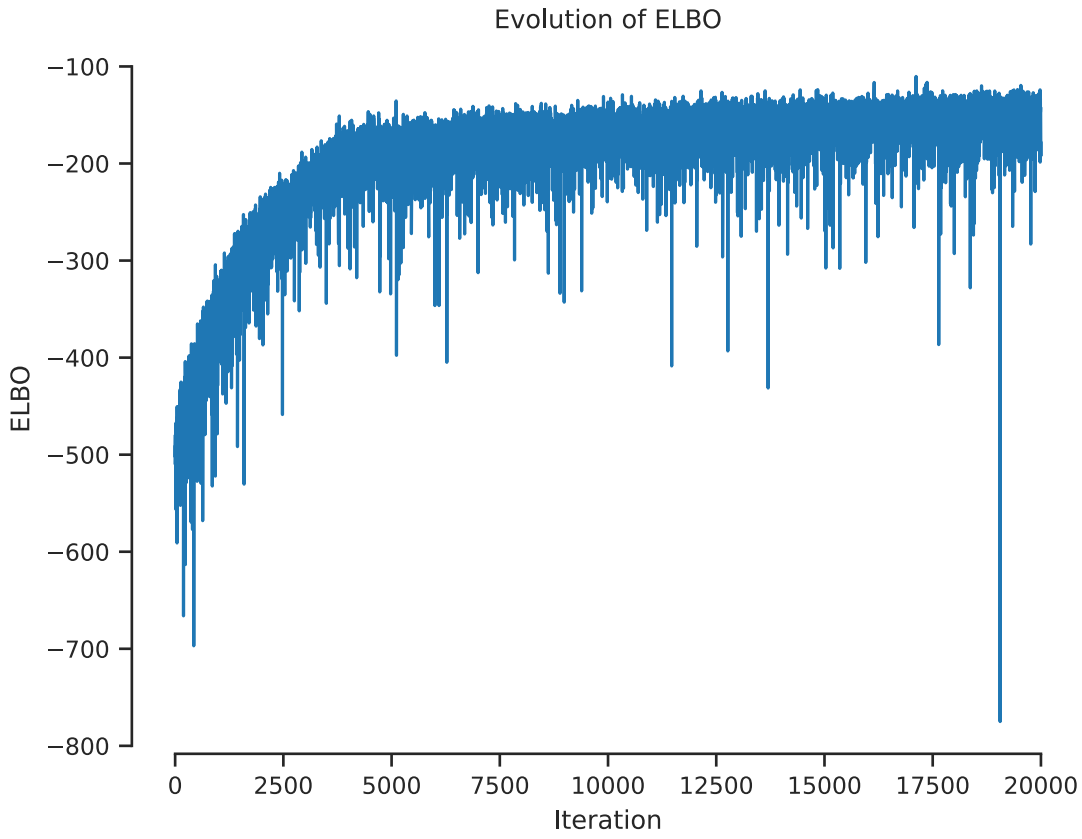
alpha ~ Exponential
w ~ Normal
alpha_noise ~ Exponential
w_noise ~ Normal
ell ~ Exponential
ell_noise ~ Exponential
y ~ Normal

Train the model using 20,000 iterations and plot the evolution of the ELBO.

[ ]:
```
num_iter = 20000
elbos, params = train(model, guide, (X, Y), num_iter=num_iter)
```

```
Iteration: 0 Loss: 502.1592712402344
Iteration: 1000 Loss: 353.76092529296875
Iteration: 2000 Loss: 254.5151824951172
Iteration: 3000 Loss: 241.0348358154297
Iteration: 4000 Loss: 183.45211791992188
Iteration: 5000 Loss: 189.0581512451172
Iteration: 6000 Loss: 346.3852233886719
Iteration: 7000 Loss: 165.91571044921875
Iteration: 8000 Loss: 160.26187133789062
Iteration: 9000 Loss: 163.69659423828125
Iteration: 10000 Loss: 148.20701599121094
Iteration: 11000 Loss: 171.71636962890625
Iteration: 12000 Loss: 195.18707275390625
Iteration: 13000 Loss: 164.0347442626953
Iteration: 14000 Loss: 150.8982391357422
Iteration: 15000 Loss: 152.4806671142578
Iteration: 16000 Loss: 150.81568908691406
Iteration: 17000 Loss: 155.5046844482422
Iteration: 18000 Loss: 148.16567993164062
Iteration: 19000 Loss: 165.37203979492188
```

[ ]:
```
# Plot the evolution of the ELBO
fig, ax = plt.subplots()
ax.plot(elbos)
ax.set(xlabel="Iteration", ylabel="ELBO", title="Evolution of ELBO")
sns.despine(trim=True);
```



Extend the model to make predictions:

[ ]:
```
def predictive_model(X, y, num_centers=50):
    params = model(X, y, num_centers)
    w = params["w"]
    w_noise = params["w_noise"]
    ell = params["ell"]
    ell_noise = params["ell_noise"]
    sigma = params["sigma"]
    x_centers = params["x_centers"]
    xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
    Phi = RadialBasisFunctions(x_centers, ell)(xs)
    Phi_noise = RadialBasisFunctions(x_centers, ell_noise)(xs)
    predictions = pyro.deterministic("predictions", Phi @ w)
```

```
    sigma = pyro.deterministic("sigma", torch.exp(Phi_noise @ w_noise))
    predictions_with_noise = pyro.sample("predictions_with_noise", dist.
 ↪Normal(predictions, sigma))
    return locals()
```

Extract the posterior predictive distribution using 10,000 samples and calculate the aleatory/epistemic uncertainty:

[ ]:
```
# Your code here
post_pred = pyro.infer.Predictive(predictive_model, guide=guide,␣
 ↪num_samples=10000)(X, Y)
# We will predict here:
xs = torch.linspace(X.min(), X.max(), 100).unsqueeze(-1)
# Extract the predictions and get the epistemic uncertainty
predictions = post_pred["predictions"]
p_500, p_025, p_975 = np.percentile(predictions, [50, 2.5, 97.5], axis=0)
# Extract predictions with noise and get the aleatory uncertainty
predictions_with_noise = post_pred["predictions_with_noise"]
ap_025, ap_975 = np.percentile(predictions_with_noise, [2.5, 97.5], axis=0)
```

Plot the data, the median, the 95% credible interval of epistemic uncertainty, the 95% credible interval of aleatory uncertainty, and five samples from the posterior:

[ ]:
```
# Plot the observed data
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(X, Y, 'kx', label="Observed Data")

# Plot the median
ax.plot(xs.flatten(), p_500.flatten(), color="r", lw=2, label="Median")

# Plot the epistemic uncertainty interval
ax.fill_between(xs.flatten(), p_025.flatten(), p_975.flatten(), color="b",
                alpha=.2, label="95% Epistemic Uncertainty Interval")

# Plot the aleatory uncertainty interval
ax.fill_between(xs.flatten(), ap_025.flatten(), ap_975.flatten(), color="g",
                alpha=.2, label="95% Aleatory Uncertainty Interval")

# Plot 5 samples from the posterior
for i in range(5):
    sample_idx = np.random.choice(predictions.shape[0])
    sample = predictions[sample_idx]
    if i == 0:
        ax.plot(xs.flatten(), sample.flatten(), '--',
                color="darkviolet", alpha=.7, label="Samples")
    else:
        ax.plot(xs.flatten(), sample.flatten(), '--',
```

```
                color="darkviolet", alpha=.7)

# Add axis labels and legend
ax.set(xlabel="$x$", ylabel="$y$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True);
```