Pattern & Anomaly Detection Lab

Experiment 9

Data Transformations(Focus on kernel Approximation and Pairwise Kernels)

Submitted By:

Dhruv Singhal

500075346

R177219074

AIML B3

Submitted To:

Dr. Gopal Phartiyal

Asst. Professor

SOCS

UPES

CODE:

```
import matplotlib.pyplot as plt
import numpy as np


def plot_gpr_samples(gpr_model, n_samples, ax):
    """Plot samples drawn from the Gaussian process model.

    If the Gaussian process model is not trained then the drawn samples are
    drawn from the prior distribution. Otherwise, the samples are drawn from
    the posterior distribution. Be aware that a sample here corresponds to a
    function. """

    x = np.linspace(0, 5, 100)
    X = x.reshape(-1, 1)

    y_mean, y_std = gpr_model.predict(X, return_std=True)
    y_samples = gpr_model.sample_y(X, n_samples)

    for idx, single_prior in enumerate(y_samples.T):
        ax.plot(
            x,
            single_prior,
            linestyle="--",
            alpha=0.7,
            label=f"Sampled function #{idx + 1}",
        )
    ax.plot(x, y_mean, color="black", label="Mean")
    ax.fill_between(
        x,
        y_mean - y_std,
        y_mean + y_std,
        alpha=0.1,
        color="black",
        label=r"$\pm$ 1 std. dev.",
    )
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_ylim([-3, 3])



rng = np.random.RandomState(4)
X_train = rng.uniform(0, 5, 10).reshape(-1, 1)
y_train = np.sin((X_train[:, 0] - 2.5) ** 2)
n_samples = 5


### RBF Kernel

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

kernel = 1.0 * RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
```

```python

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Radial Basis Function kernel", fontsize=18)
plt.tight_layout()


print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)




### Rational Quadratic Kernel

from sklearn.gaussian_process.kernels import RationalQuadratic

kernel = 1.0 * RationalQuadratic(length_scale=1.0, alpha=0.1, alpha_bounds=(1e-5, 1e15))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Rational Quadratic kernel", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
```

```
#%%
### ExpSineSquared


from sklearn.gaussian_process.kernels import ExpSineSquared


kernel = 1.0 * ExpSineSquared(length_scale=1,periodicity=1,length_scale_bounds=(1e-5, 1e15),periodicity_bounds=(1e-5, 1e15))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("ExpSineSquared", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%
### Matern


from sklearn.gaussian_process.kernels import Matern


kernel = 1.0 * Matern(length_scale=1,length_scale_bounds=(1e-5, 1e15),nu=1.5)
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Matern", fontsize=18)
plt.tight_layout()
```

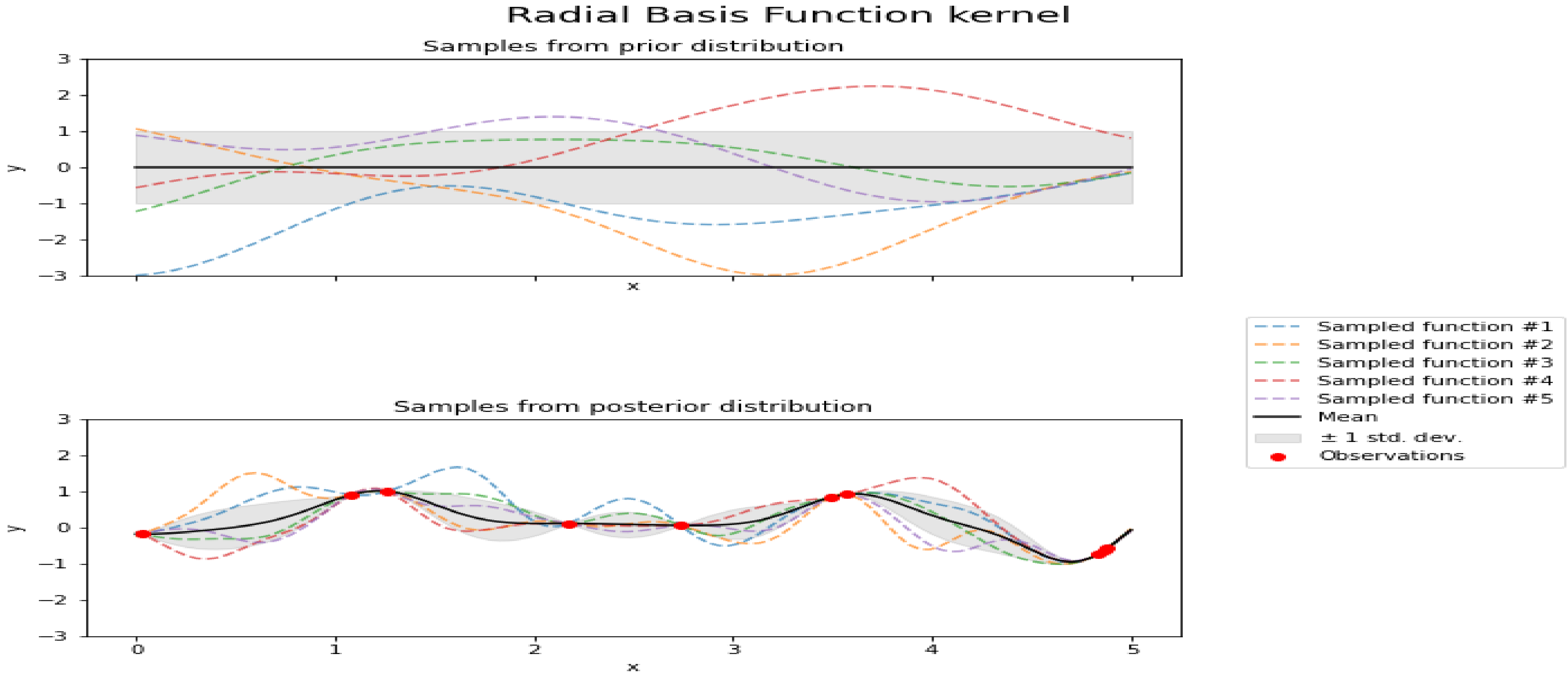
```python

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)

#%%
### WhiteKernel


from sklearn.gaussian_process.kernels import WhiteKernel


kernel = 1.0 * WhiteKernel(noise_level=1,noise_level_bounds=(1e-5, 1e15))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("WhiteKernel", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)

#%%

### Exponentiation


from sklearn.gaussian_process.kernels import Exponentiation


kernel = 1.0 * Exponentiation(RationalQuadratic(), exponent=2)
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
```

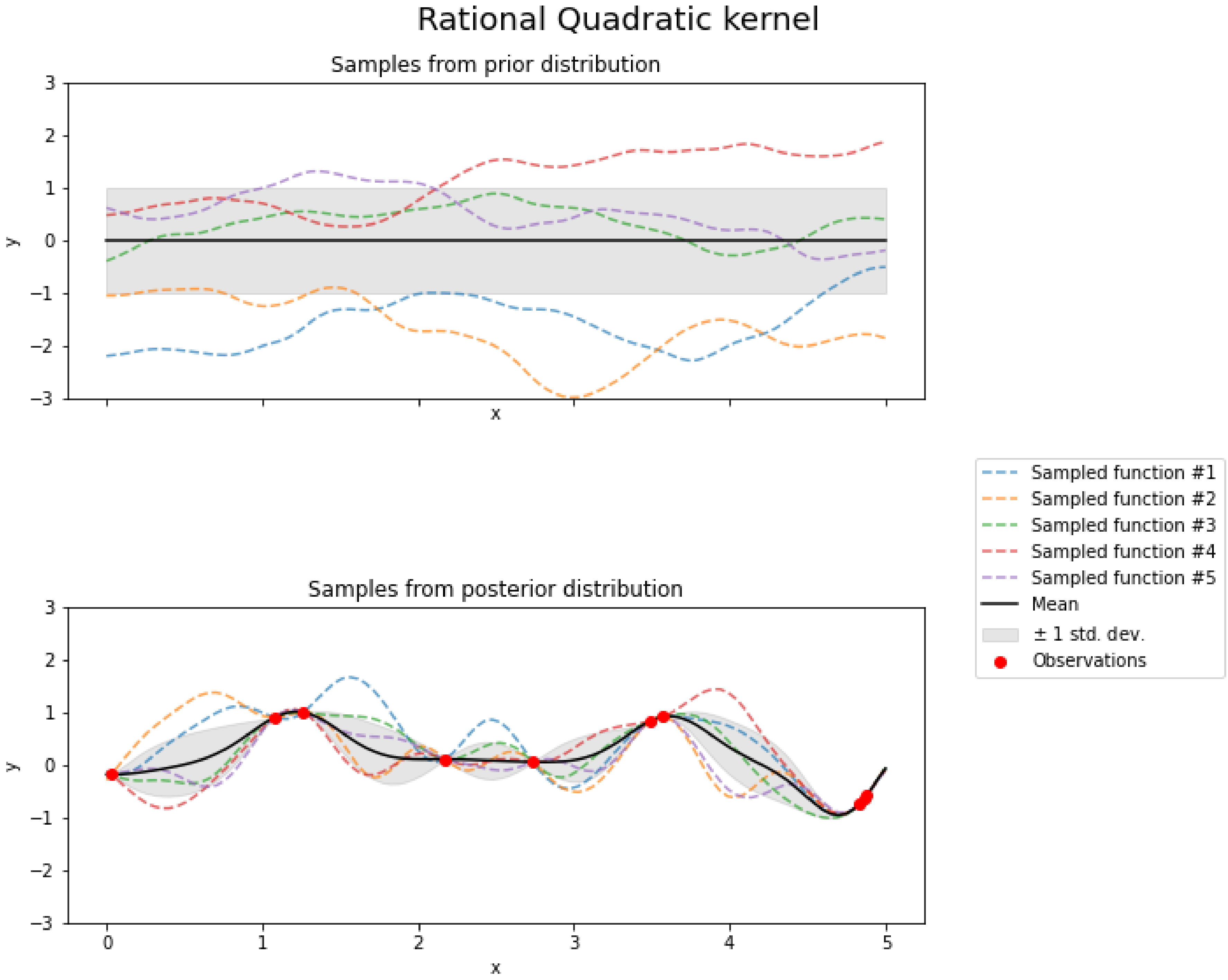
```python

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Exponentiation", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%
### DotProduct


from sklearn.gaussian_process.kernels import DotProduct


kernel = 1.0 * DotProduct(sigma_0=1,sigma_0_bounds=(1e-5, 1e5))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("DotProduct", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%
### Sum kernel


from sklearn.gaussian_process.kernels import Sum


kernel = 1.0 * Sum(DotProduct(),WhiteKernel())
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
```

```python

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Sum kernel", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%

```

OUTPUT:

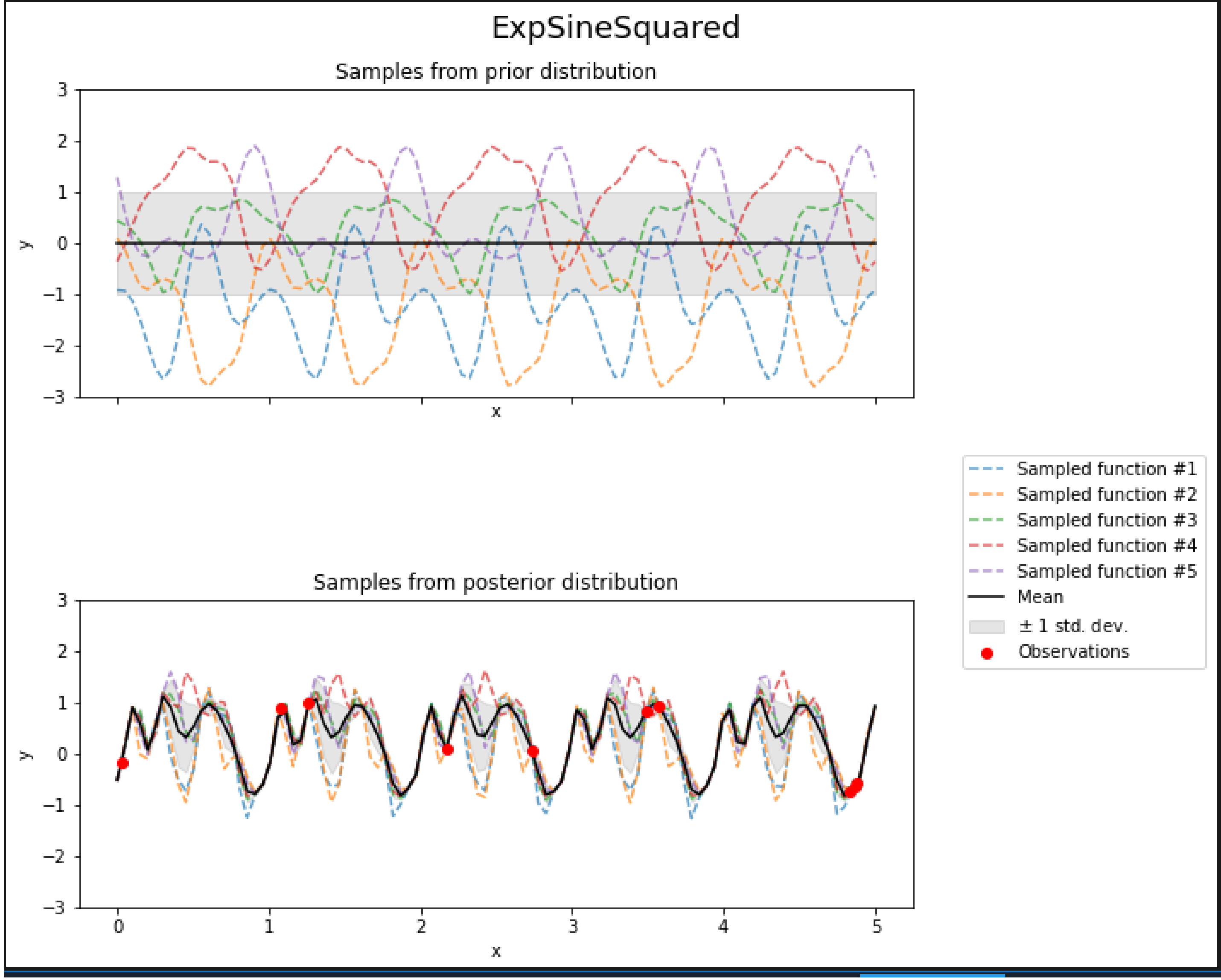
```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * RBF(length_scale=1))+++
Kernel parameters after fit:
0.594**2 * RBF(length_scale=0.279)
Log-likelihood: -0.067
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\_gpr.py:506: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  _check_optimize_result("lbfgs", opt_res)
Kernel parameters before fit:
1**2 * RationalQuadratic(alpha=0.1, length_scale=1))
Kernel parameters after fit:
0.594**2 * RationalQuadratic(alpha=1.78e+06, length_scale=0.279)
Log-likelihood: -0.067
```



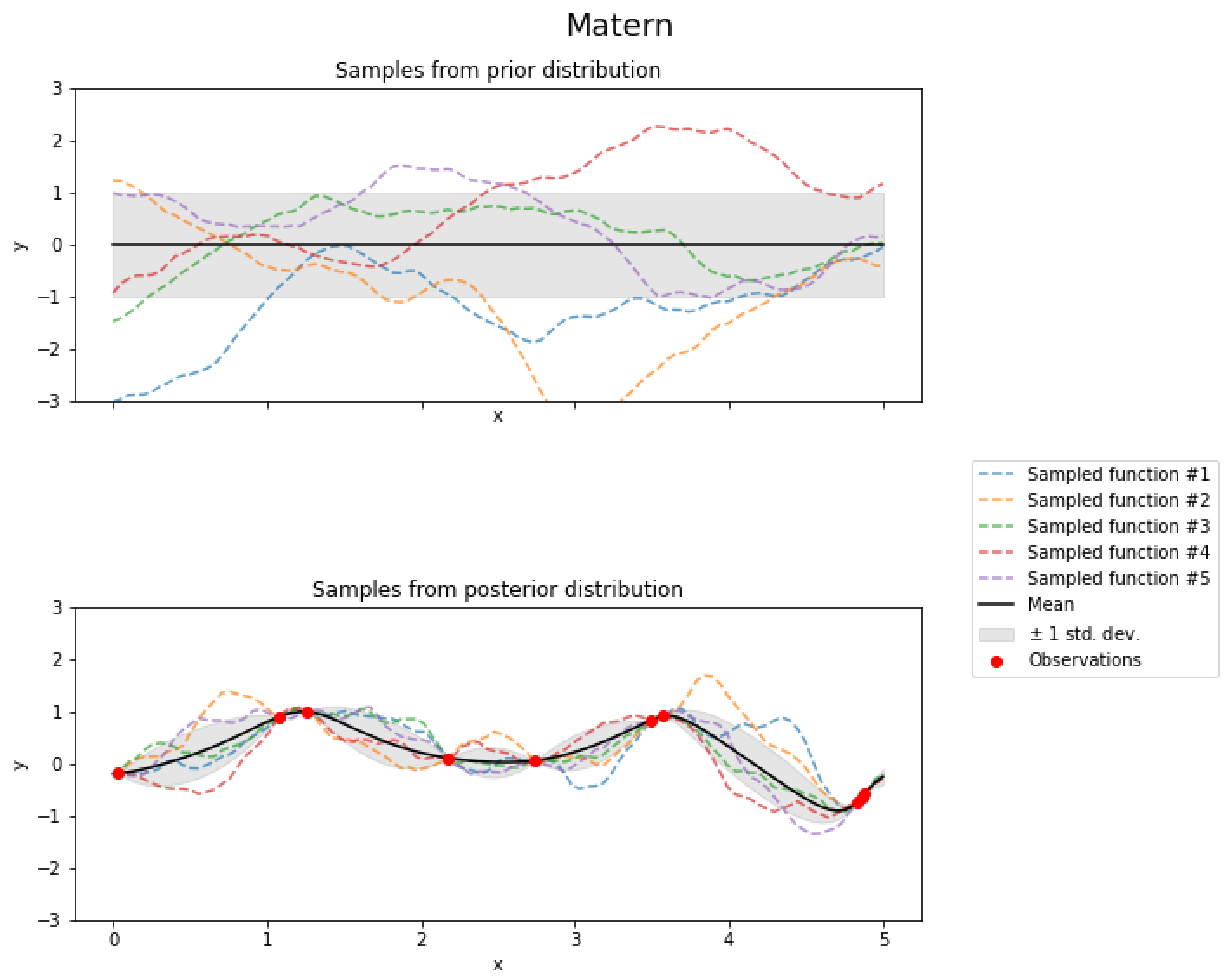


Rational Quadratic kernel
Samples from prior distribution
Samples from posterior distribution
y
x
3
2
1
0
−1
−2
−3
0
1
2
3
4
5
Sampled function #1
Sampled function #2
Sampled function #3
Sampled function #4
Sampled function #5
Mean
± 1 std. dev.
Observations

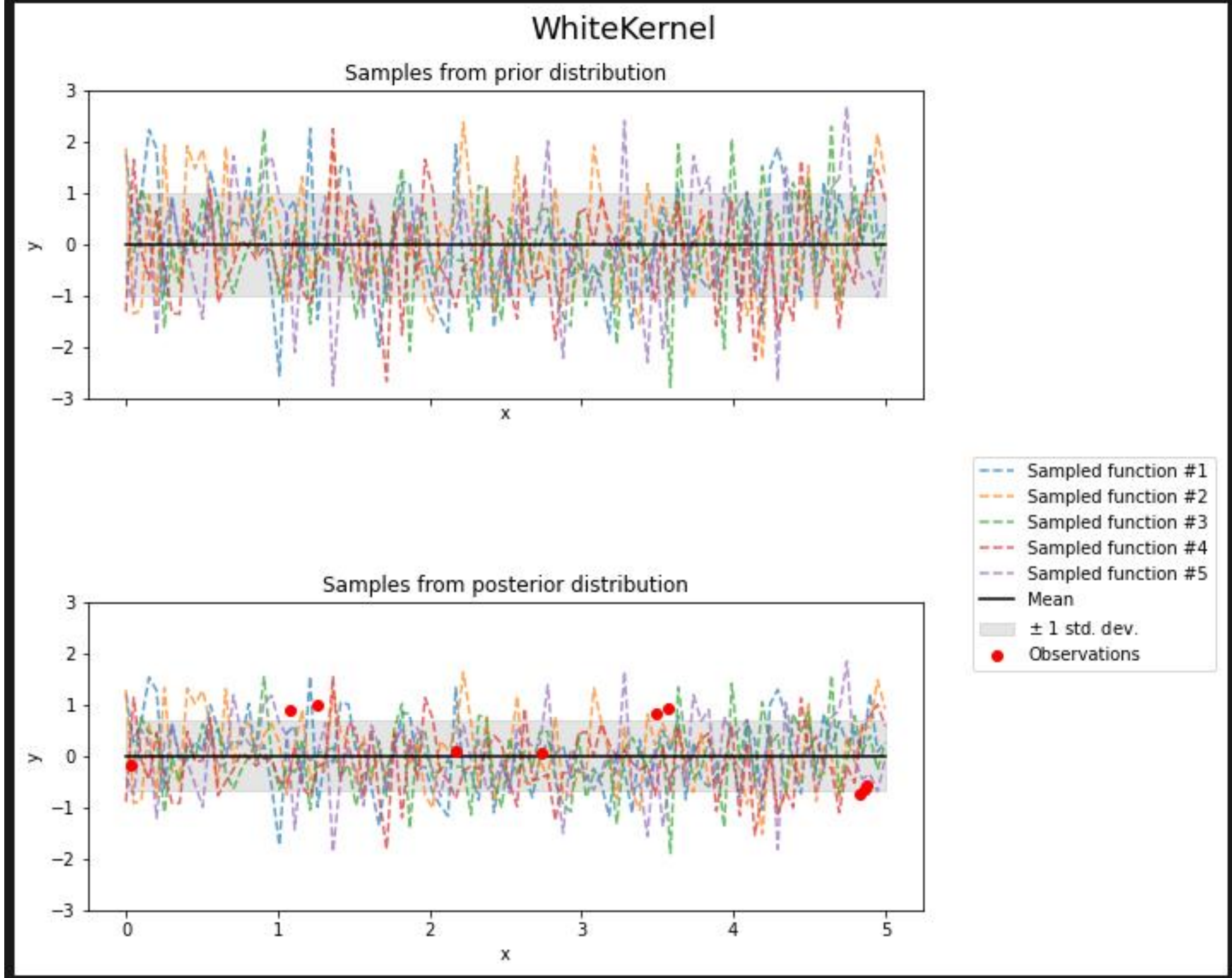
```
In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * ExpSineSquared(length_scale=1, periodicity=1))
Kernel parameters after fit:
0.736**2 * ExpSineSquared(length_scale=0.438, periodicity=0.978)
Log-likelihood: -5.478
```



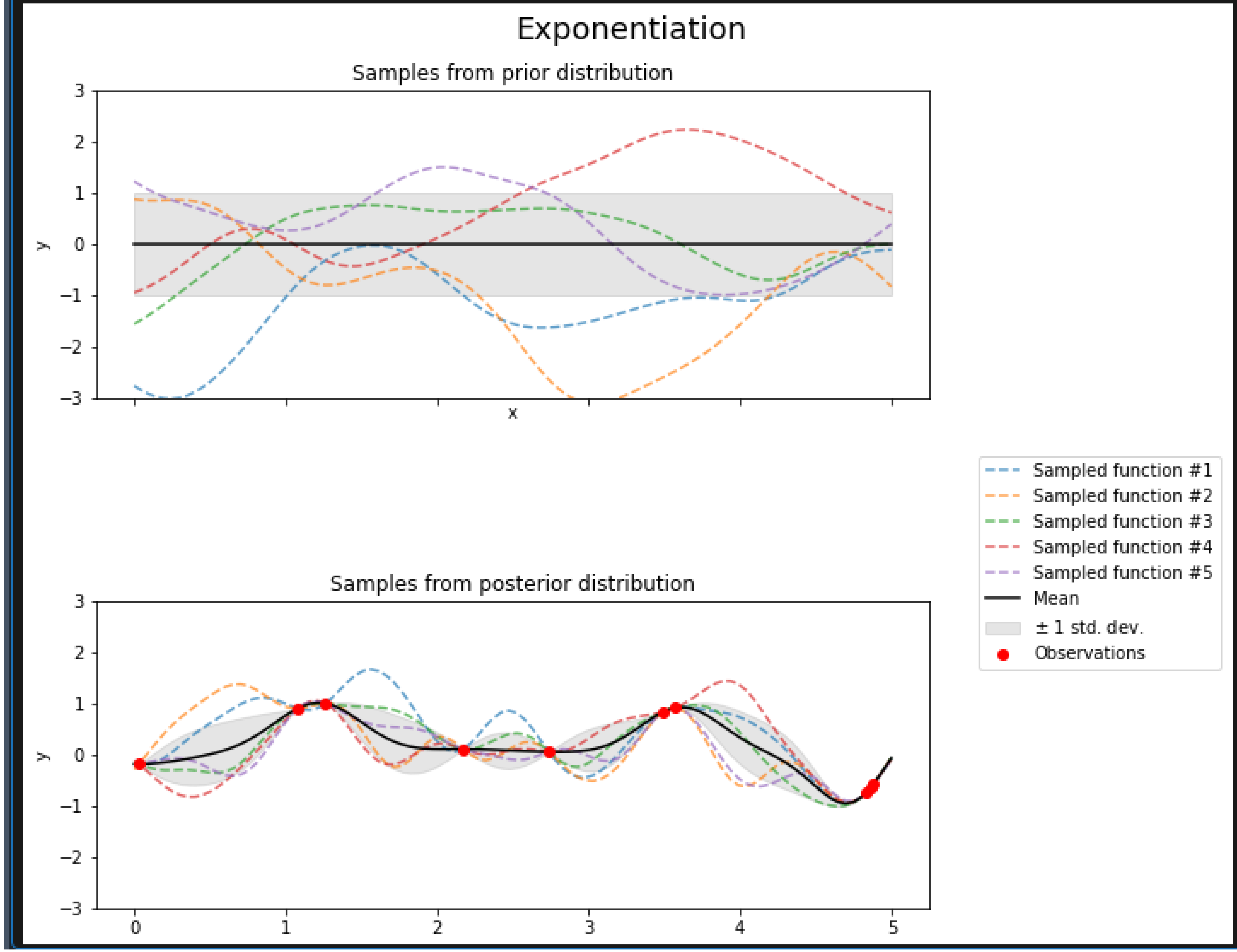
```
In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * Matern(length_scale=1, nu=1.5))
Kernel parameters after fit:
0.609**2 * Matern(length_scale=0.484, nu=1.5)
Log-likelihood: -1.185
```



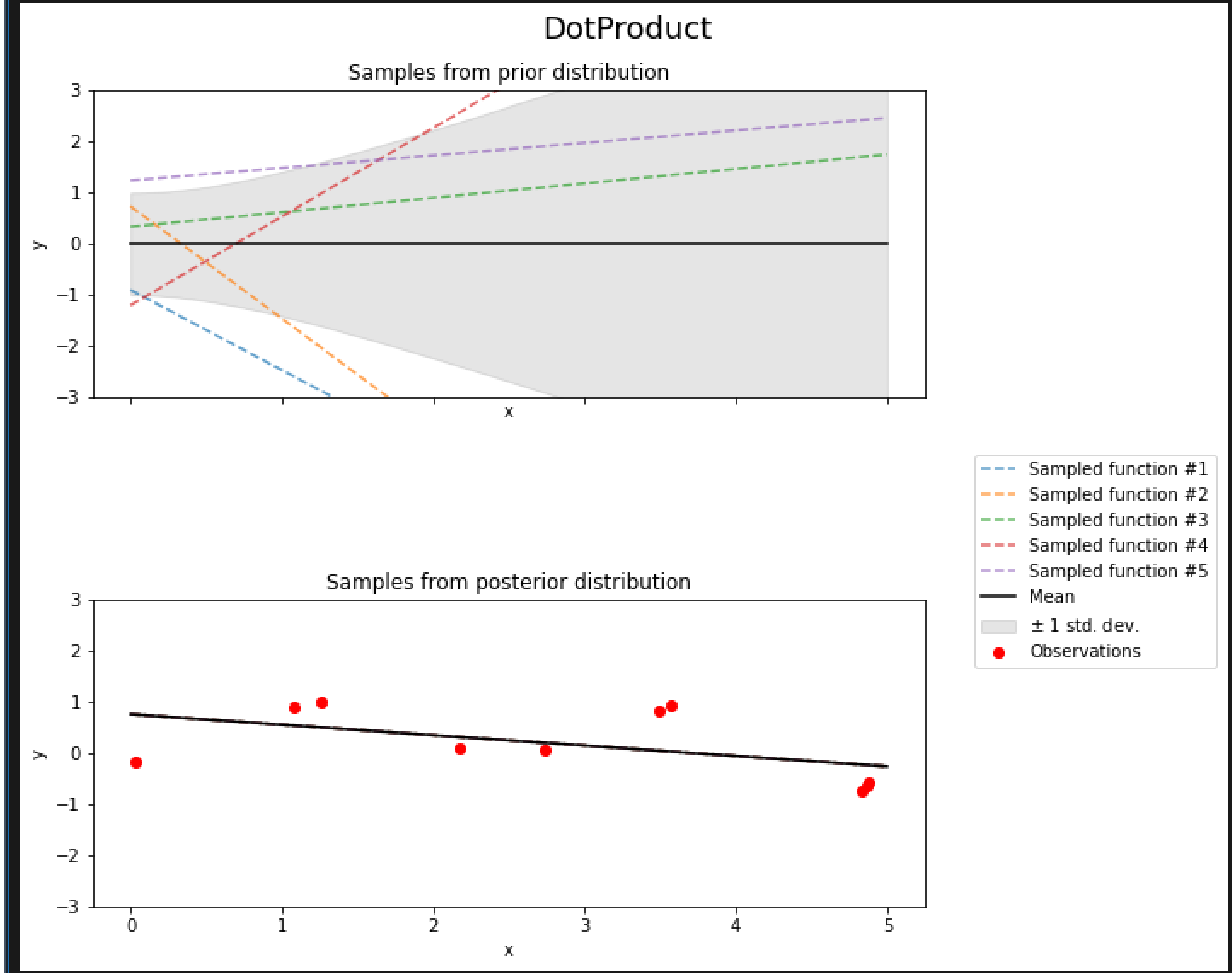
```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * WhiteKernel(noise_level=1))
Kernel parameters after fit:
0.827**2 * WhiteKernel(noise_level=0.684)
Log-likelihood: -10.386
```



```
In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/exp9.py')
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\kernels.py:418: ConvergenceWarning: The optimal value found for dimension 0 of
parameter k2__kernel__alpha is close to the specified upper bound 100000.0. Increasing the bound and calling fit again may find a better value.
  ConvergenceWarning)
Kernel parameters before fit:
1**2 * RationalQuadratic(alpha=1, length_scale=1) ** 2)
Kernel parameters after fit:
0.594**2 * RationalQuadratic(alpha=1e+05, length_scale=0.394) ** 2
Log-likelihood: -0.067
```



```
In [6]: runcell(5, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * DotProduct(sigma_0=1))
Kernel parameters after fit:
0.999**2 * DotProduct(sigma_0=0.999)
Log-likelihood: -16319044829.997
```



```
In [7]: runcell(6, 'B:/3rd year/5th sem/P&AD/exp9.py')
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\kernels.py:409: ConvergenceWarning: The optimal value found for dimension 0 of
parameter k1__constant_value is close to the specified lower bound 1e-05. Decreasing the bound and calling fit again may find a better value.
  ConvergenceWarning)
Kernel parameters before fit:
1**2 * DotProduct(sigma_0=1) + WhiteKernel(noise_level=1))
Kernel parameters after fit:
0.00316**2 * DotProduct(sigma_0=0.339) + WhiteKernel(noise_level=4.67e+04)
Log-likelihood: -10.387
```