

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:

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4
5 def plot_gpr_samples(gpr_model, n_samples, ax):
6     """Plot samples drawn from the Gaussian process model.
7
8     If the Gaussian process model is not trained then the drawn samples are
9     drawn from the prior distribution. Otherwise, the samples are drawn from
10    the posterior distribution. Be aware that a sample here corresponds to a
11    function. """
12
13    x = np.linspace(0, 5, 100)
14    X = x.reshape(-1, 1)
15
16    y_mean, y_std = gpr_model.predict(X, return_std=True)
17    y_samples = gpr_model.sample_y(X, n_samples)
18
19    for idx, single_prior in enumerate(y_samples.T):
20        ax.plot(
21            x,
22            single_prior,
23            linestyle="--",
24            alpha=0.7,
25            label=f"Sampled function #{idx + 1}",
26        )
27    ax.plot(x, y_mean, color="black", label="Mean")
28    ax.fill_between(
29        x,
30        y_mean - y_std,
31        y_mean + y_std,
32        alpha=0.1,
33        color="black",
34        label=r"$\pm$ 1 std. dev.",
35    )
36    ax.set_xlabel("x")
37    ax.set_ylabel("y")
38    ax.set_ylim([-3, 3])
39
40
41
42 rng = np.random.RandomState(4)
43 X_train = rng.uniform(0, 5, 10).reshape(-1, 1)
44 y_train = np.sin((X_train[:, 0] - 2.5) ** 2)
45 n_samples = 5
46
47
48 ### RBF Kernel
49
50 from sklearn.gaussian_process import GaussianProcessRegressor
51 from sklearn.gaussian_process.kernels import RBF
52
53 kernel = 1.0 * RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))
54 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
55
56 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
```

```

57
58 # plot prior
59 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[0])
60 axes[0].set_title("Samples from prior distribution")
61
62 # plot posterior
63 gpr.fit(X_train, y_train)
64 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[1])
65 axes[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
66 axes[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
67 axes[1].set_title("Samples from posterior distribution")
68
69 fig.suptitle("Radial Basis Function kernel", fontsize=18)
70 plt.tight_layout()
71
72
73 print(f"Kernel parameters before fit:\n{kernel}")
74 print(
75     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
76     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
77 )
78
79
80
81
82 #### Rational Quadratic Kernel
83
84 from sklearn.gaussian_process.kernels import RationalQuadratic
85
86 kernel = 1.0 * RationalQuadratic(length_scale=1.0, alpha=0.1, alpha_bounds=(1e-5, 1e15))
87 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
88
89 fig, axes = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
90
91 # plot prior
92 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[0])
93 axes[0].set_title("Samples from prior distribution")
94
95 # plot posterior
96 gpr.fit(X_train, y_train)
97 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[1])
98 axes[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
99 axes[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
100 axes[1].set_title("Samples from posterior distribution")
101
102 fig.suptitle("Rational Quadratic kernel", fontsize=18)
103 plt.tight_layout()
104
105 print(f"Kernel parameters before fit:\n{kernel}")
106 print(
107     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
108     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
109 )

```

```

110 #####
111 ### ExpSineSquared
112
113
114 from sklearn.gaussian_process.kernels import ExpSineSquared
115
116
117 kernel = 1.0 * ExpSineSquared(length_scale=1,periodicity=1,length_scale_bounds=(1e-5, 1e15),periodicity_bounds=(1e-5, 1e15))
118 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
119
120 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
121
122 # plot prior
123 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
124 axs[0].set_title("Samples from prior distribution")
125
126 # plot posterior
127 gpr.fit(X_train, y_train)
128 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
129 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
130 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
131 axs[1].set_title("Samples from posterior distribution")
132
133 fig.suptitle("ExpSineSquared", fontsize=18)
134 plt.tight_layout()
135
136 print(f"Kernel parameters before fit:\n{kernel}")
137 print(
138     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
139     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
140 )
141 #####
142 ### Matern
143
144
145 from sklearn.gaussian_process.kernels import Matern
146
147
148 kernel = 1.0 * Matern(length_scale=1,length_scale_bounds=(1e-5, 1e15),nu=1.5)
149 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
150
151 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
152
153 # plot prior
154 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
155 axs[0].set_title("Samples from prior distribution")
156
157 # plot posterior
158 gpr.fit(X_train, y_train)
159 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
160 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
161 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
162 axs[1].set_title("Samples from posterior distribution")
163
164 fig.suptitle("Matern", fontsize=18)
165 plt.tight_layout()

```

```

166
167 print(f"Kernel parameters before fit:\n{kernel}")
168 print(
169     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
170     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
171 )
172
173 ##%
174 ### WhiteKernel
175
176
177 from sklearn.gaussian_process.kernels import WhiteKernel
178
179
180 kernel = 1.0 * WhiteKernel(noise_level=1, noise_level_bounds=(1e-5, 1e15))
181 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
182
183 fig, axes = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
184
185 # plot prior
186 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[0])
187 axes[0].set_title("Samples from prior distribution")
188
189 # plot posterior
190 gpr.fit(X_train, y_train)
191 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[1])
192 axes[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
193 axes[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
194 axes[1].set_title("Samples from posterior distribution")
195
196 fig.suptitle("WhiteKernel", fontsize=18)
197 plt.tight_layout()
198
199 print(f"Kernel parameters before fit:\n{kernel}")
200 print(
201     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
202     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
203 )
204
205 ##%
206
207 ### Exponentiation
208
209
210 from sklearn.gaussian_process.kernels import Exponentiation
211
212
213 kernel = 1.0 * Exponentiation(RationalQuadratic(), exponent=2)
214 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
215
216 fig, axes = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
217
218 # plot prior
219 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[0])
220 axes[0].set_title("Samples from prior distribution")
221
222 # plot posterior

```

```

221
222 # plot posterior
223 gpr.fit(X_train, y_train)
224 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[1])
225 axes[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
226 axes[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
227 axes[1].set_title("Samples from posterior distribution")
228
229 fig.suptitle("Exponentiation", fontsize=18)
230 plt.tight_layout()
231
232 print(f"Kernel parameters before fit:\n{kernel}")
233 print(
234     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
235     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
236 )
237 ###
238 ### DotProduct
239
240
241 from sklearn.gaussian_process.kernels import DotProduct
242
243
244 kernel = 1.0 * DotProduct(sigma_0=1, sigma_0_bounds=(1e-5, 1e5))
245 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
246
247 fig, axes = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
248
249 # plot prior
250 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[0])
251 axes[0].set_title("Samples from prior distribution")
252
253 # plot posterior
254 gpr.fit(X_train, y_train)
255 plot_gpr_samples(gpr, n_samples=n_samples, ax=axes[1])
256 axes[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
257 axes[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
258 axes[1].set_title("Samples from posterior distribution")
259
260 fig.suptitle("DotProduct", fontsize=18)
261 plt.tight_layout()
262
263 print(f"Kernel parameters before fit:\n{kernel}")
264 print(
265     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
266     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
267 )
268 ###
269 ### Sum kernel
270
271
272 from sklearn.gaussian_process.kernels import Sum
273
274
275 kernel = 1.0 * Sum(DotProduct(), WhiteKernel())
276 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

```

```

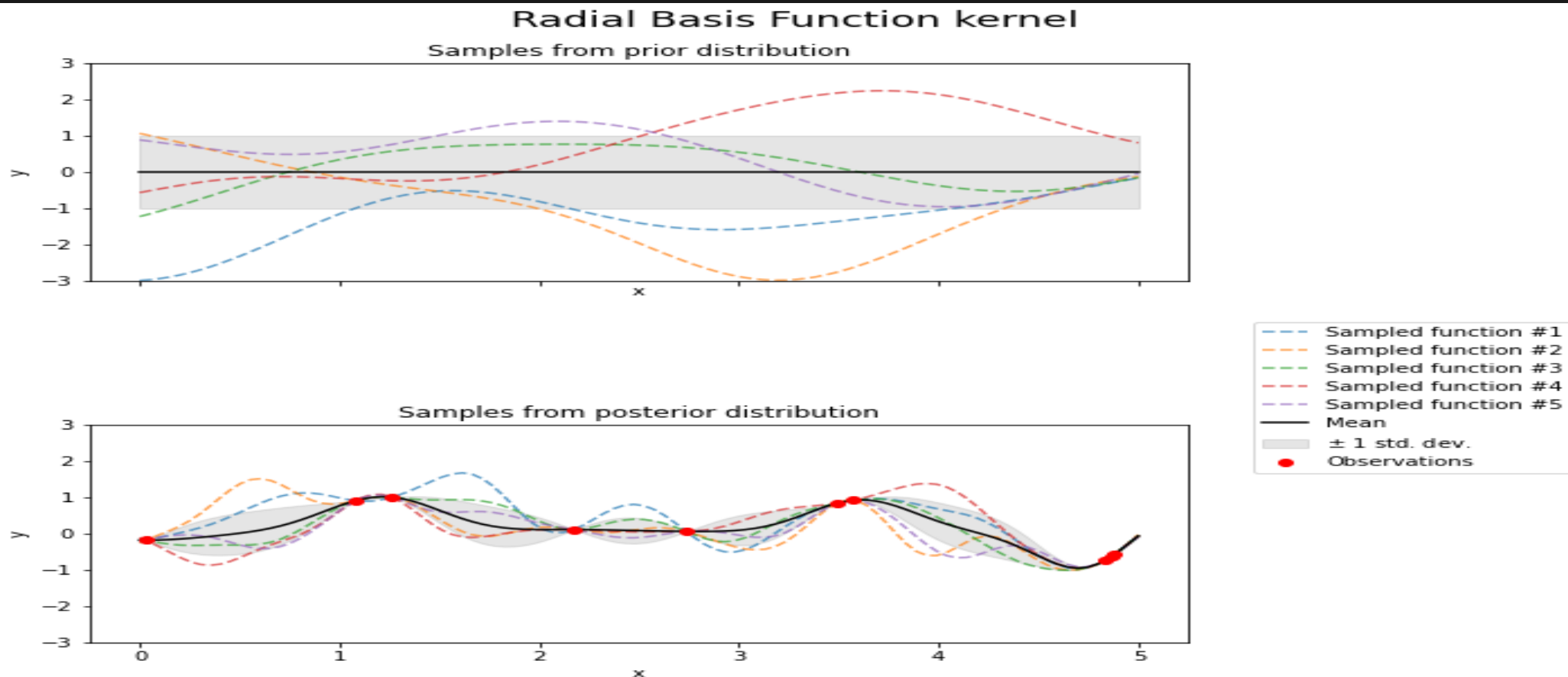
277
278 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
279
280 # plot prior
281 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
282 axs[0].set_title("Samples from prior distribution")
283
284 # plot posterior
285 gpr.fit(X_train, y_train)
286 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
287 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
288 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
289 axs[1].set_title("Samples from posterior distribution")
290
291 fig.suptitle("Sum kernel", fontsize=18)
292 plt.tight_layout()
293
294 print(f"Kernel parameters before fit:\n{kernel}")
295 print(
296     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
297     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
298 )
299 ##0%
300

```

OUTPUT:

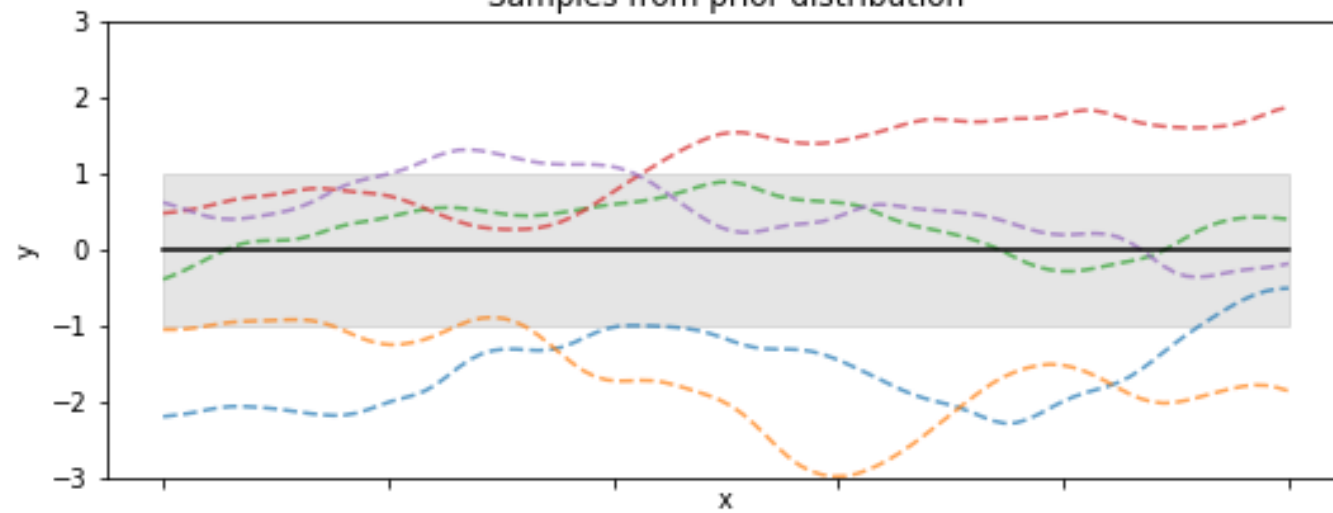
```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * RBF(length_scale=1)+++
Kernel parameters after fit:
0.594**2 * RBF(length_scale=0.279)
Log-likelihood: -0.067
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\_gpr.py:506: ConvergenceWarning: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

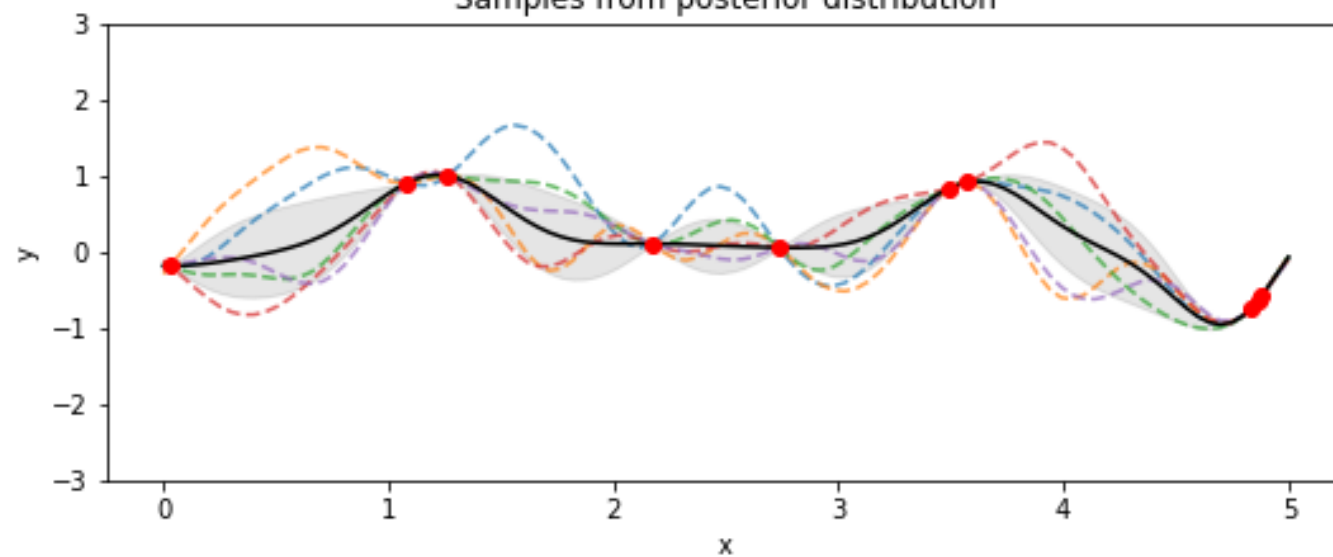


Rational Quadratic kernel

Samples from prior distribution

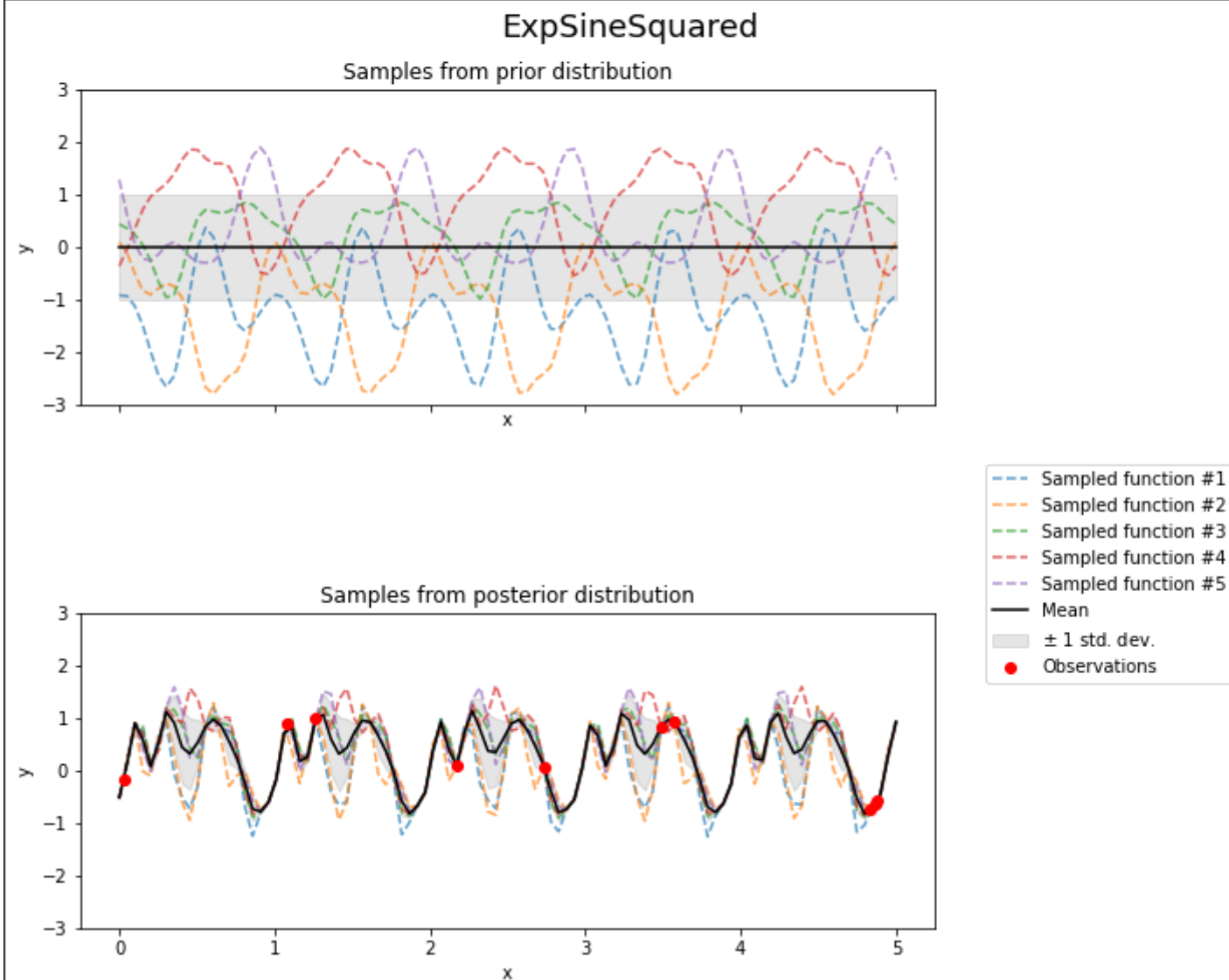


Samples from posterior distribution

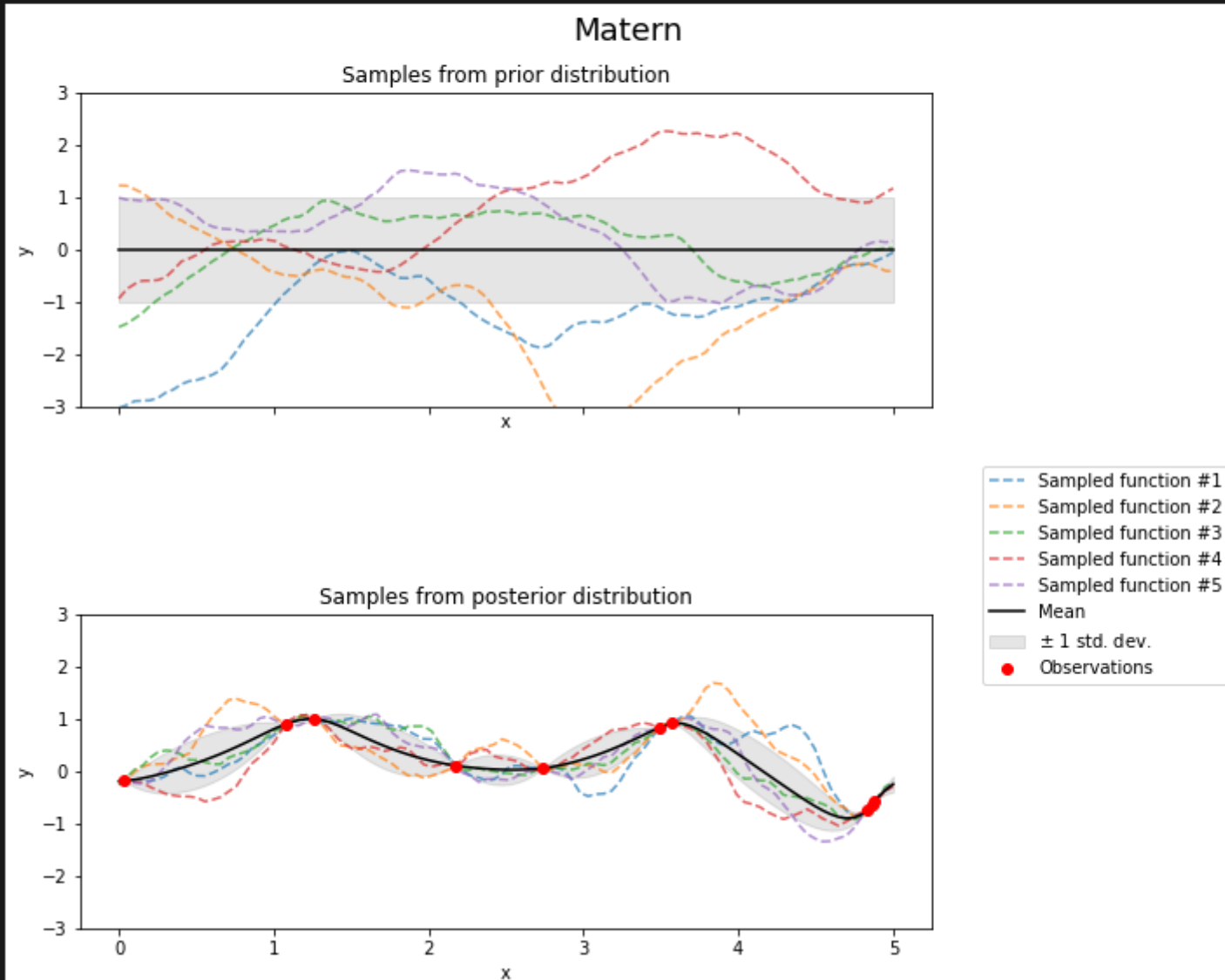


- Sampled function #1
- Sampled function #2
- Sampled function #3
- Sampled function #4
- Sampled function #5
- Mean
- ± 1 std. dev.
- Observations

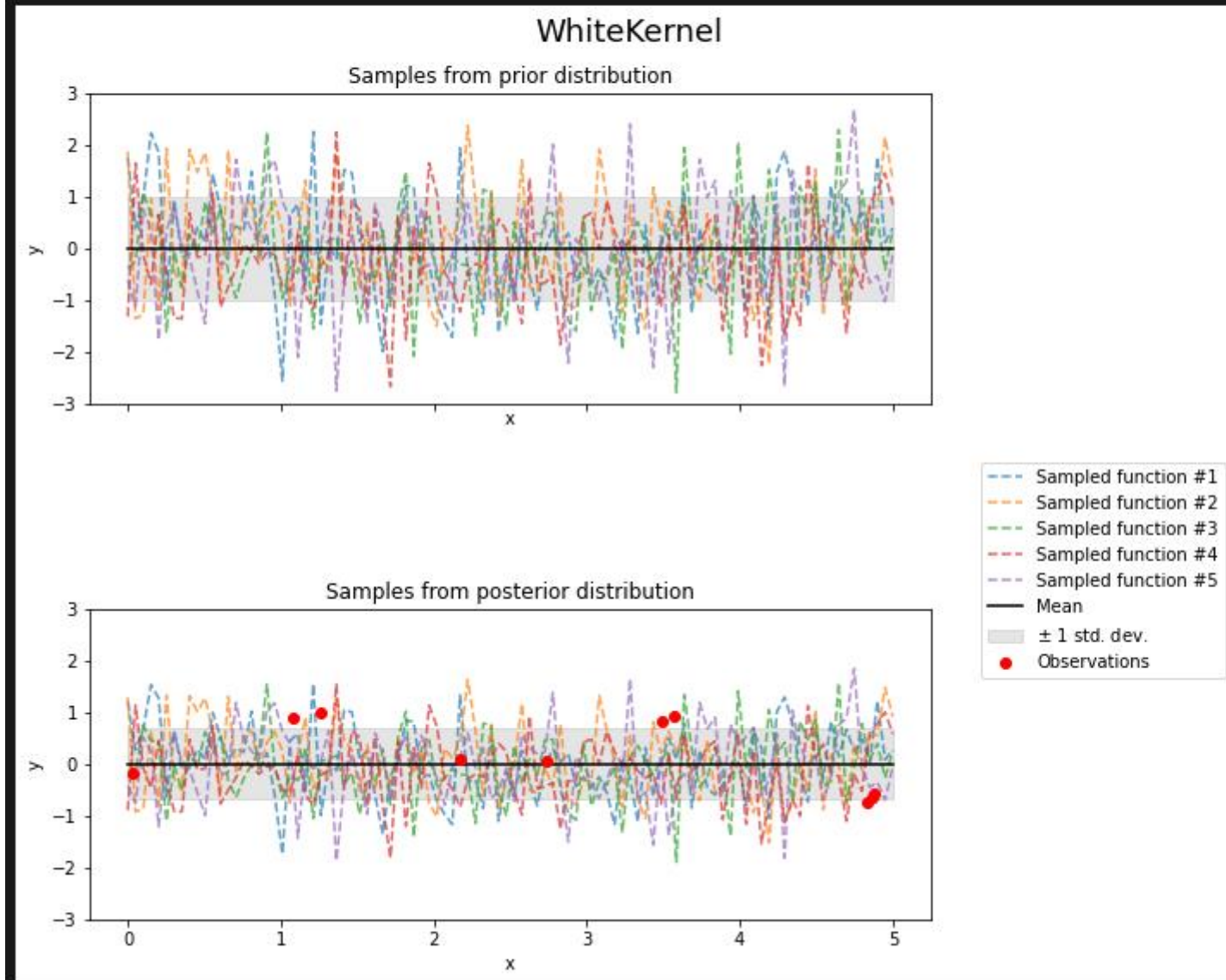
```
In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1*2 * ExpSineSquared(length_scale=1, periodicity=1))
Kernel parameters after fit:
0.736*2 * ExpSineSquared(length_scale=0.438, periodicity=0.978)
Log-likelihood: -5.478
```



```
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)
```

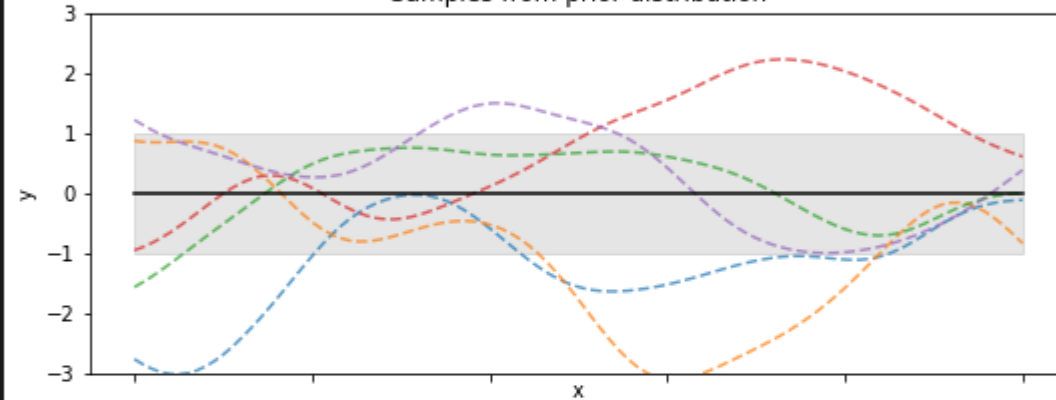
```
Kernel parameters after fit:
```

```
0.594**2 * RationalQuadratic(alpha=1e+05, length_scale=0.394) ** 2
```

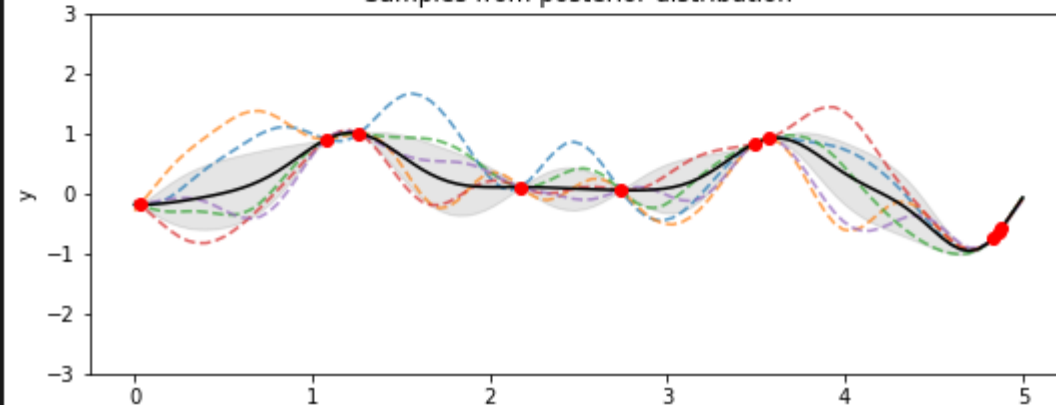
```
Log-likelihood: -0.067
```

Exponentiation

Samples from prior distribution



Samples from posterior distribution



- Sampled function #1
- Sampled function #2
- Sampled function #3
- Sampled function #4
- Sampled function #5
- Mean
- ± 1 std. dev.
- Observations

```
In [6]: runcell(5, 'B:/3rd year/5th sem/P&AD/exp9.py')
```

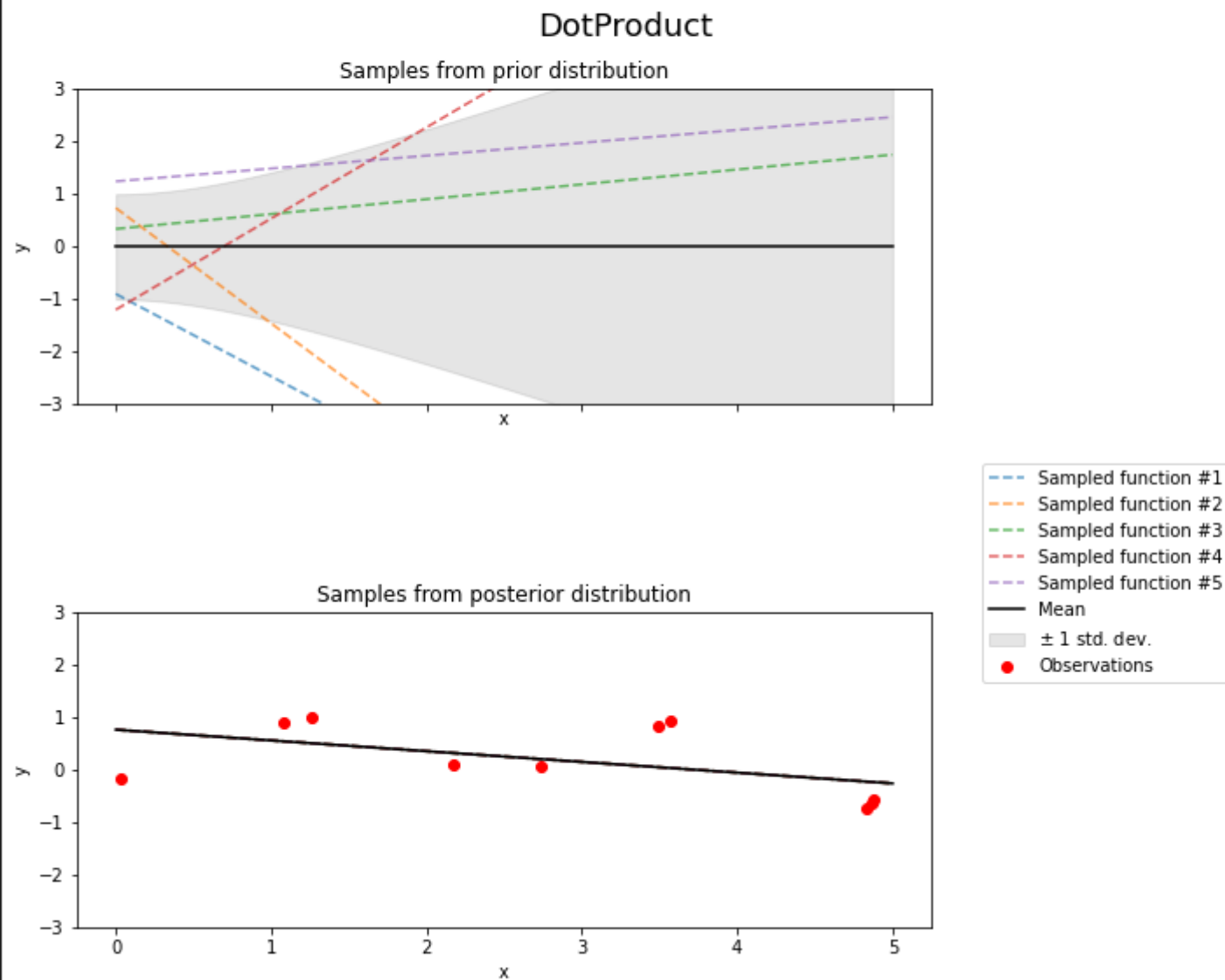
Kernel parameters before fit:

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
```

