Bayesian Optimization

Peyman Kor

8/9/2021

In this post I will be explaining the step-by-step process to perfom Bayesin Optimization (BO). Well, first let's define the optimization problem in simple mathematics and then I will folow it with BO.

Optimization

In math term, the optimization can be defined as:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) \\ \text{subject to} \quad & x \subseteq \chi \end{aligned} \tag{1}$$

where in above equation x, is a input value. Let's say you doing hyperparameter optimization, and x could be number of layers in deep learning model or could be number of trees in random forest model. $f(x)$ is the objective function that you want to maximize. In the case of hyperparamter optimization, generally it could be RMS.

Now, going forward, the question is while we have many optimization methods, why we should use Bayesian Optimization? Actually it is good question, based on author experience, if the optimization problem in hand, has following properties, BO is well suited:

- f is explicitly unknown. In another term, f is a "balck-box" function.
- The surface of f is multi-modal. Meaning that f is non-convex in the domain of χ , and the optimization algorithm must visit all local optima to find the "global" one.
- Most importantly, forward evaluation of f is computationally expensive. This point will be discussed more in detail below.

Optimzaton problem: What we are Optimizing?

In order to have a better explanation and visulization of the steps in BO, here I considred a 1D function. The aim is to find global optimum of this function.

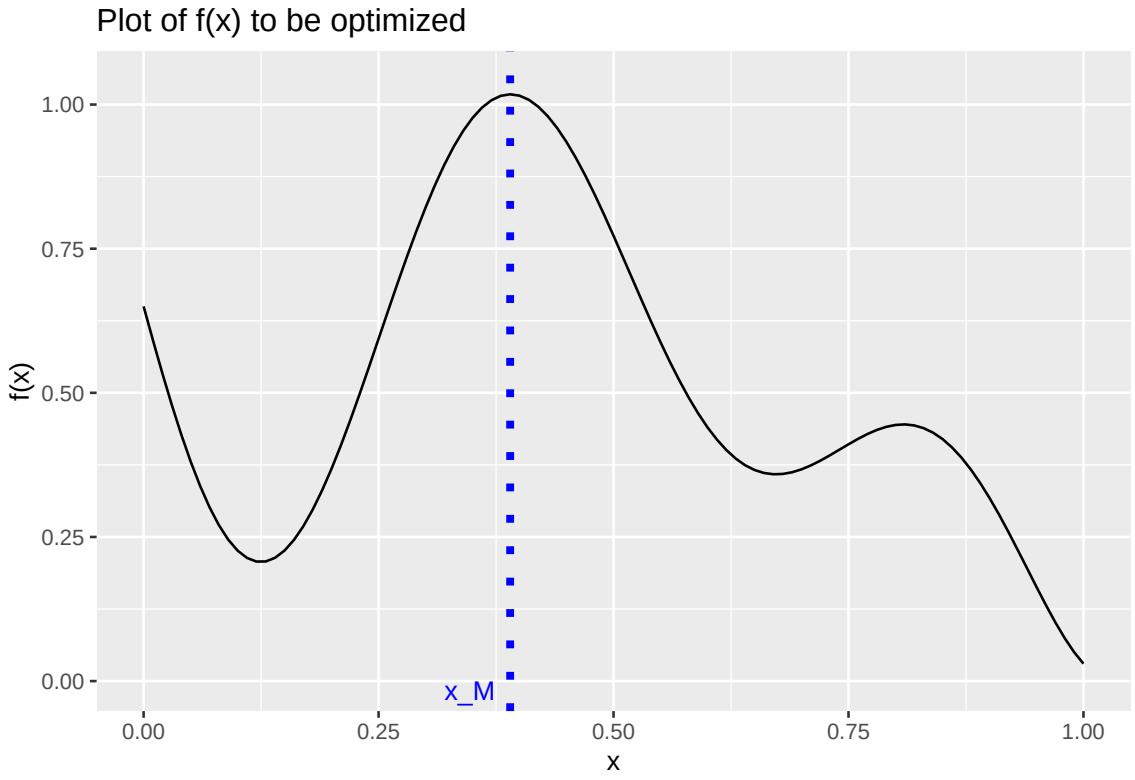
$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) = 1 - \frac{1}{2}\left(\frac{\sin(12x)}{1+x} + 2\cos(7x)x^5 + 0.7\right) \\ \text{subject to} \quad & 0 \leq x \leq 1 \end{aligned} \tag{2}$$

Since the analytical expression of function is available and being a 1-D problem, we know beforehand that the global optimum of function is $x_M = 3.90$. We plotted the function below. Notice that the function in the plot has some local optimum around $x = 0.75$. I porpusfully selected a 1-D problem with a non-convex structure, in order to see whether BO avoids local optima and converges to a global one or not.

```r
# library for plotting and data manipulation
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
# 1d function defined above
one_d_fun <- function(x) {

  y <- (1-1/2*((sin(12*x)/(1+x))+(2*cos(7*x)*x^5)+0.7))

  return(y)
}

# domain of plot
x_domain <- seq(0,1,0.01)

# building matrix of x and y
df_x_y <- function(x) {

  y <- one_d_fun(x)

  df <- tibble(x,y)

  }

# making domain of x and y
data_domain <- df_x_y(x_domain)

# visualization
ggplot(data_domain, aes(x,y)) +
  geom_line() +
  geom_vline(xintercept = 0.390, linetype="dotted",
                color = "blue", size=1.5) +
  annotate("text", x=0.32, y=0, label= "x_M", hjust = 0, vjust=+1 ,colour = "blue") +
  ylim(0,1.04) +
  ylab("f(x)") +
  ggtitle("Plot of f(x) to be optimized")
```




```
ggsave("output_plots/truth_func.png", width = 20, height = 10, units = "cm", dpi=600)
```

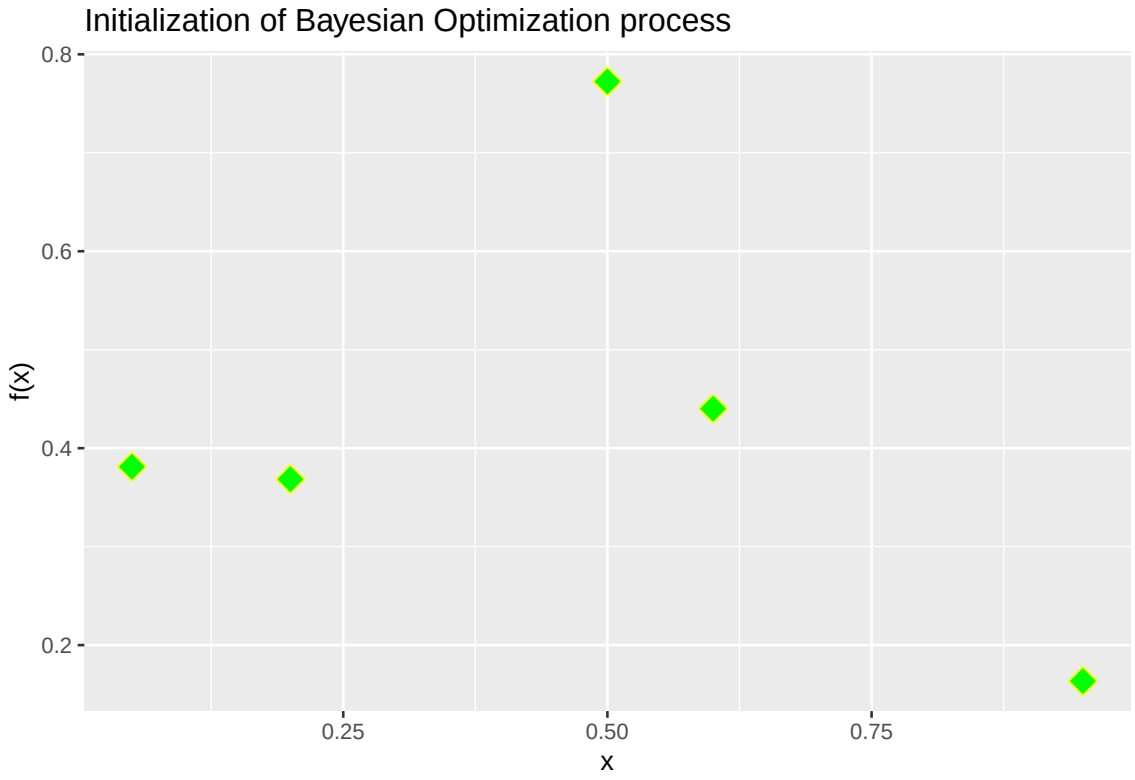
However, know that the exact analytical expression of the objective function in many real-world problems is not available (black-box optimization). What is available is a *sample* of X and $f(X)$, represented as $\mathcal{D} = [X, f(X)]$. Therefore, in the 1-D example, in order to mimic the real world case, we sample a few points to form our $\mathcal{D}$. We know the analytical expression of the objective function and global optimum of the objective function in hindsight, just for the case we want to compare the performance of BO workflow :).

To form $\mathcal{D} = [X, f(X)]$, a sample of five points, $X = [0.05, 0.2, 0.5, 0.6, 0.95]$ was selected to initialize the workflow. This $f(X)$ vector with their correspondent $f(X)$, forms the

$$\mathcal{D} = [X, f_X] = [[0.05, 0.2, 0.5, 0.6, 0.95]; [0.38, 0.36, 0.77, 0.44, 0.16]]$$

.

```
# define seed for reproducibility
set.seed(123)

# initial observation, X
X <- c(0.05,0.2,0.5,0.6,0.95)
obs_data <- df_x_y(X)

# plot the initial X
ggplot(data=obs_data) +
  geom_point(aes(x=x,y=y),fill="green", color="yellow",shape=23, size=4) +
  xlab("x") +
  ylab("f(x)")+
  ggtitle("Initialization of Bayesian Optimization process")
```




```
ggsave("output_plots/points.png", width = 20, height = 10, units = "cm", dpi=600)
```

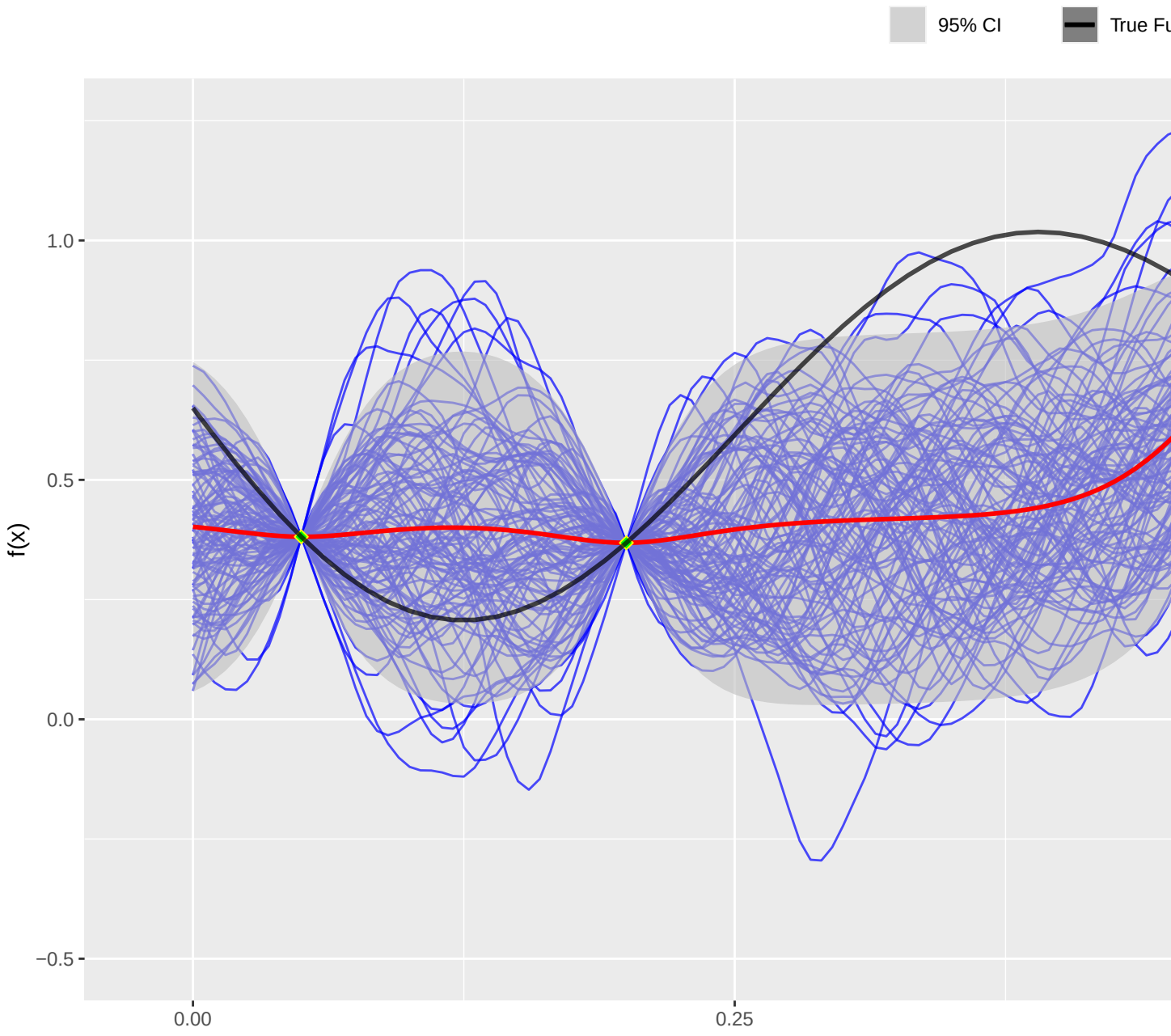
Bayesian Optimization - How it works?

Build probabilistic Model out of initial points?

Now, notice that we have five x values and five corrponding y value. Then, having this we can build a probalistic model out of this point. What that means? it means if know the $f(x)$ values for five points, we can know on some degree about other points. Without going into detail, let's build a probalistic model about the full surface of objective function, given what we know so far:

```
# we need this package to build probabilistic model (GP)
library(DiceKriging)

# Here, we are making GP model
gp_model <- function(obs_data, predict_x) {

  model <- km(~ 0, design = data.frame(x=obs_data$x), response = obs_data$y,
              multistart = 100, control =list(trace=FALSE))

  p_SK <- predict(model, newdata=data.frame(x=predict_x), type="SK",cov.compute = TRUE)

  return(list(predict_list=p_SK, cov_par=model@covariance@range.val))
}


# we define where are the prediction point, we are interested
```

```r
x_predict <- seq(0,1,0.005)

# we ask gp_model to predict at these points
predict_list <- gp_model(obs_data,x_predict)

# some hlp function for visualization
source("plot_func/plot_post_indi.R")

# viualize the posterio
posterior_1 <- plot_post_indi(predict_list$predict_list,x_predict,obs_data)

posterior_1
```




```
ggsave("output_plots/gp_model.png", width = 20, height = 10, units = "cm", dpi=600)
```

Let's make couple of observation from above figure:

- The first point to infer from the upper plot at Figure @ref(fig:exampleshow) is that there is no uncertainty on the points in $\mathcal{D}$. The reason for this is (as was discussed in the previous section), here we consider "noise-free" observations
- Also, worth mentioning that we have a wider grey area (more uncertainty) in the areas that are more distant from the observations, simply meaning uncertainty is less in points close to observation points.
- When it comes to "extrapolation", meaning in the areas outside of the range of observation points, the probabilistic model shows interesting behavior on those "extreme" area (say for example two points at

$x = 0$ and $x = 1$), the mean curve tend to move toward the mean of all observation points , here it is $f(X) = 0.42$. Suggesting the model shows the mean-reversion behavior when it comes to extrapolation.

How to find the next point to query from the expensive function?

So , we have a a probalistic model build using the 5 points above. The next question is, how to pick the next x^{next} that we should feed to our blackbox function $f(x)$ and get new (x^{next}, y^{next}). To achieve this point, we need to find the new x, x^{next} which has the maximum value of the following term:

$$\alpha_{EI}(\mathbf{x}; \theta, \mathcal{D}) = (\mu_{\mathbf{x}} - \mathbf{f}^{+} - \epsilon)\Phi(\gamma(\mathbf{x})) + \sigma_{\mathbf{x}}\phi(\gamma(\mathbf{x}))$$

Let's break up the above equation:

- $$\mu_x$$

  is the mean value from the above probalistic model at point. Lets say you want to find μ at point 0.75, then

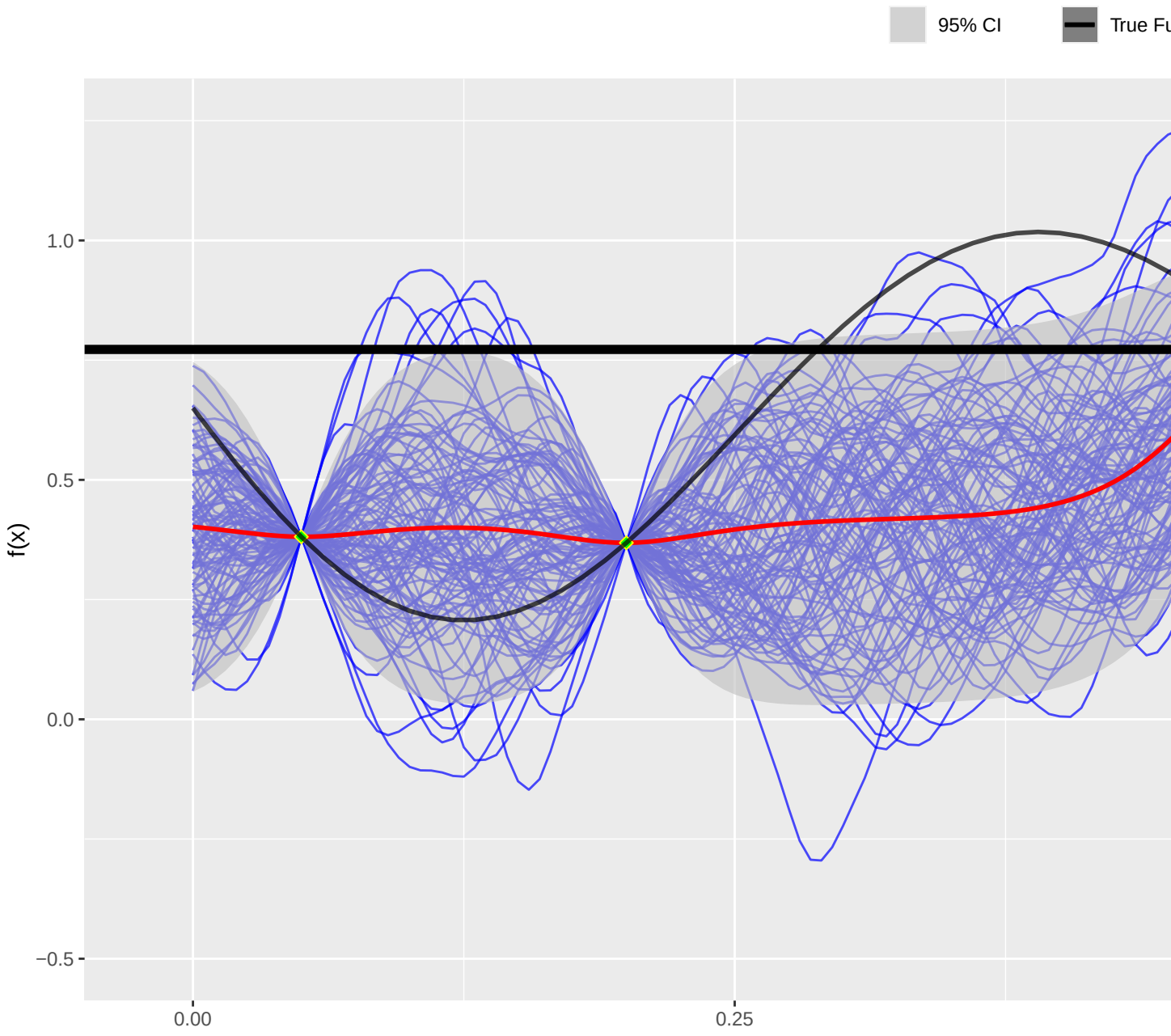
$$\mu_{x=0.75} =$$

```
# gp model forecast at x equal to 0.75
gp_model_0.75 <- gp_model(obs_data,0.75)
gp_model_0.75$predict_list$mean
```

```
## [1] 0.4183717
```

- f^{+} is the best point $\mathcal{D}$ so far. In our example, we can draw a horizontal line of the our best point so far:

```
posterior_1 +
  geom_hline(yintercept = max(obs_data$y), size=2)
```




```
ggsave("output_plots/gp_model_hline.png", width = 20, height = 10, units = "cm", dpi=600)
```

- σ_x is a standard deviation of the model each point. Lets say you want to find

$$\sigma_x$$

at point 0.75, then

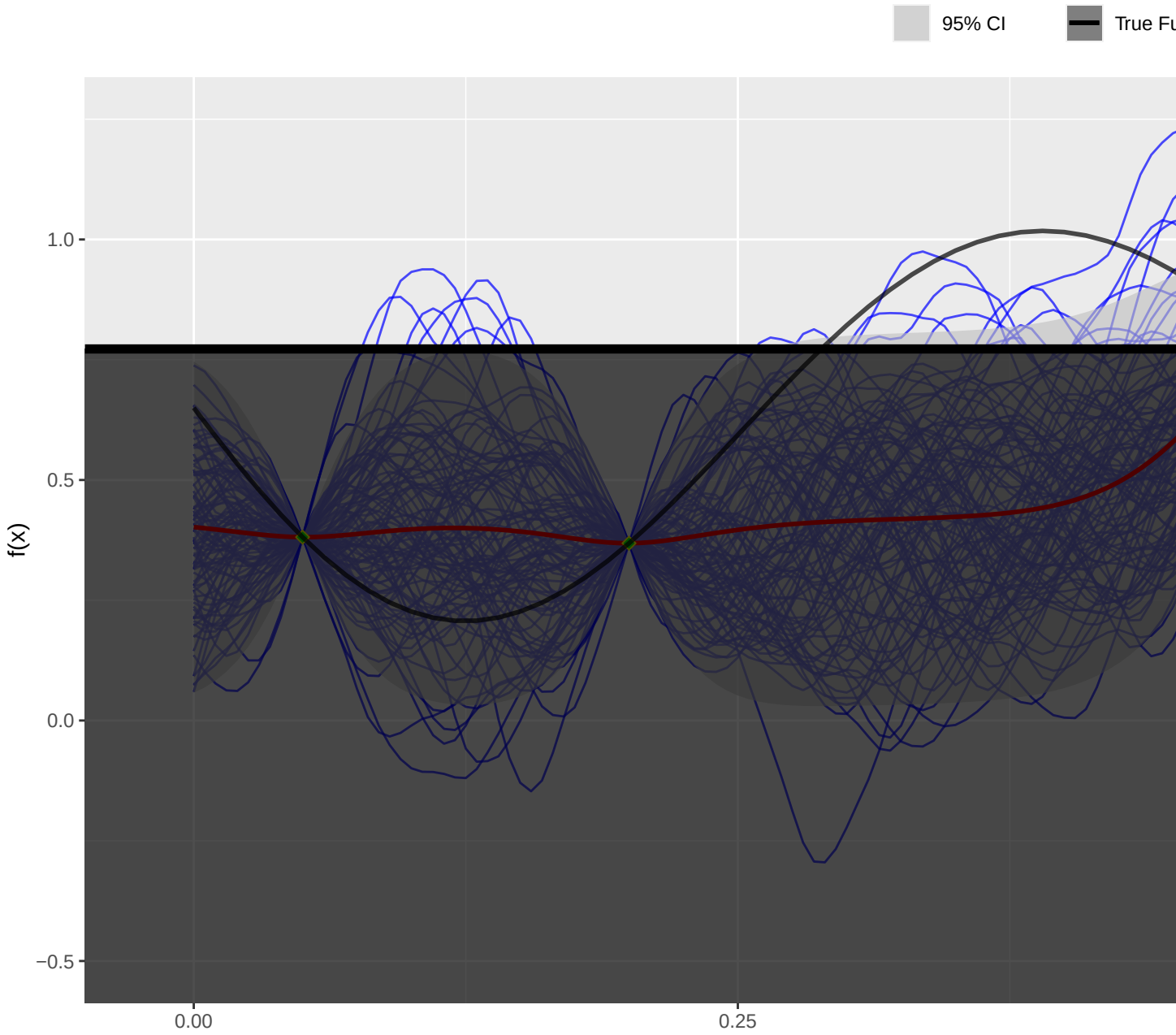
$$\sigma_{x=0.75}:$$

```
#standard deviation of gp model at x=0.75
gp_model_0.75$predict_list$sd
```

```
## [1] 0.1971042
```

Intutively, the equation says that choose the point which has the more blue lines above the blacl areas, lets say we show it with:

```
posterior_1 +
  geom_hline(yintercept = max(obs_data$y), size=2) +
  annotate("rect", xmin = -Inf, xmax = Inf, ymin = -Inf, ymax = max(obs_data$y),
           fill = "black", alpha = 0.7, color = NA)
```



```
ggsave("output_plots/gp_model_hline_black.png", width = 20, height = 10, units = "cm", dpi=600)
```

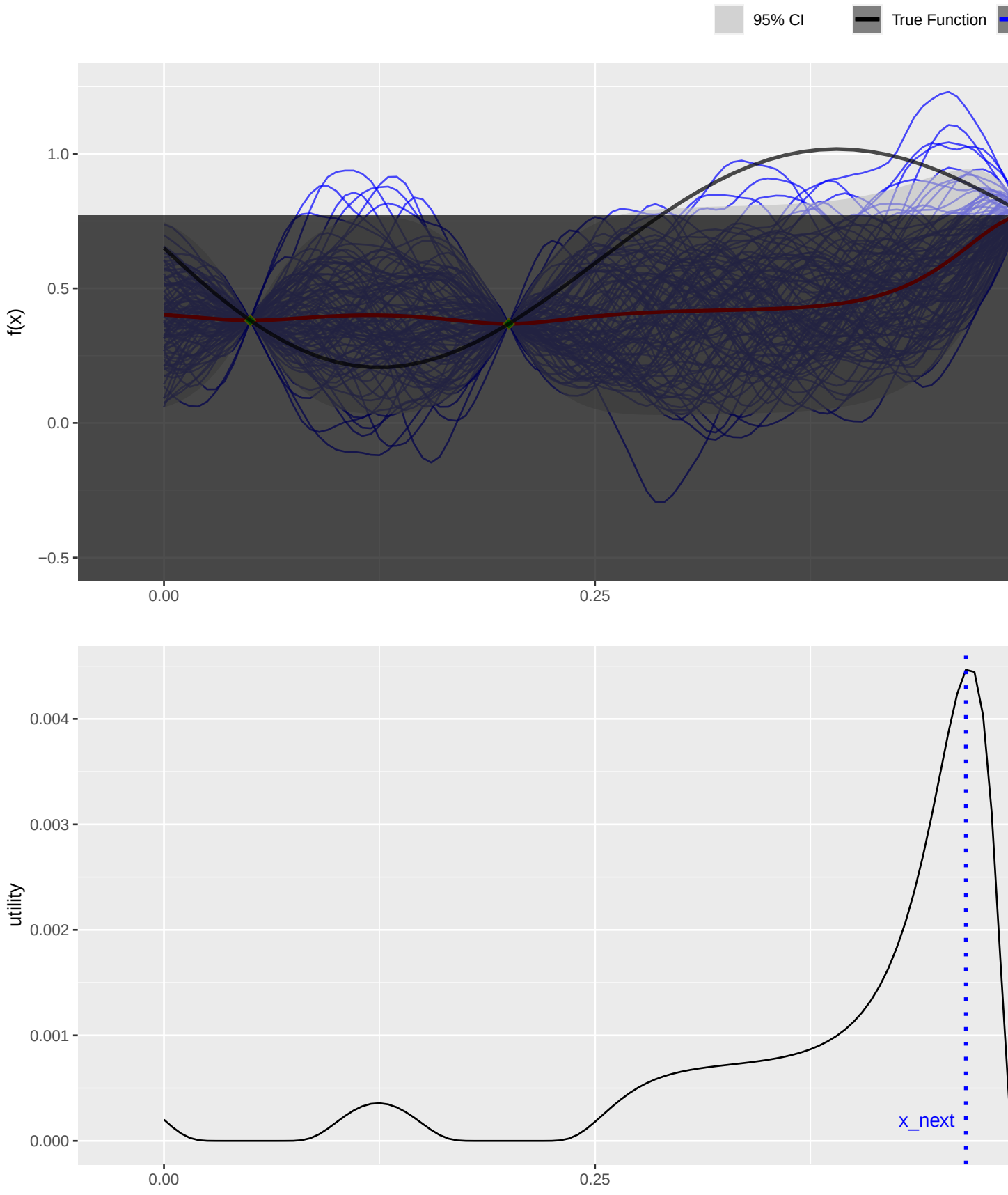
At least on my eye, the point around 0.4 has the most blue lines above the black area. But, let calculate that mathematically:

```
# function to calculate the utility (acquistion function) at each x

utility_cal <- function(predict_list, x_predict,obs_data,eps) {

  y_max <- max(obs_data$y)

  z <- (predict_list$mean - y_max - eps) / (predict_list$sd)

  utility <- (predict_list$mean - y_max - eps) * pnorm(z) + (predict_list$sd) * dnorm(z)

  new_x <- x_predict[which(utility==max(utility))]

  return(new_x)
}

new_x_point1 <- utility_cal(predict_list$predict_list,x_predict,obs_data,0.1)
new_x_point1
```

```
## [1] 0.465
```

The next point is 0.456.

```
library(patchwork)

source("plot_func/utility_cal_plot.R")
source("plot_func/utility_cal_plot_ind.R")

utility_1 <- utility_cal_plot_ind(predict_list$predict_list,
                                  x_predict,obs_data,0.1, new_x_point1)

posterior_1_ano <- posterior_1 + annotate("rect", xmin = -Inf,
                                          xmax = Inf, ymin = -Inf,
                                          ymax = max(obs_data$y),
                                          fill = "black", alpha = 0.7,
                                          color = NA)

posterior_1_ano /
  utility_1
```


95% CI
True Function
f(x)
1.0
0.5
0.0
−0.5
0.00
0.25
utility
0.004
0.003
0.002
0.001
0.000
x_next

```
ggsave("output_plots/gp_model_plus_utility.png", width = 20, height = 10, units = "cm", dpi=600)
```

The blue vertical dotted line shows the $x = 0.46$ which is the point where the utility function, is maximum. Then this x^{next} is feed into the true objective function in @ref(eq:1deq), and the pair of $[(x^{next}, x^{next}]$ is added to the initial data set, leaving

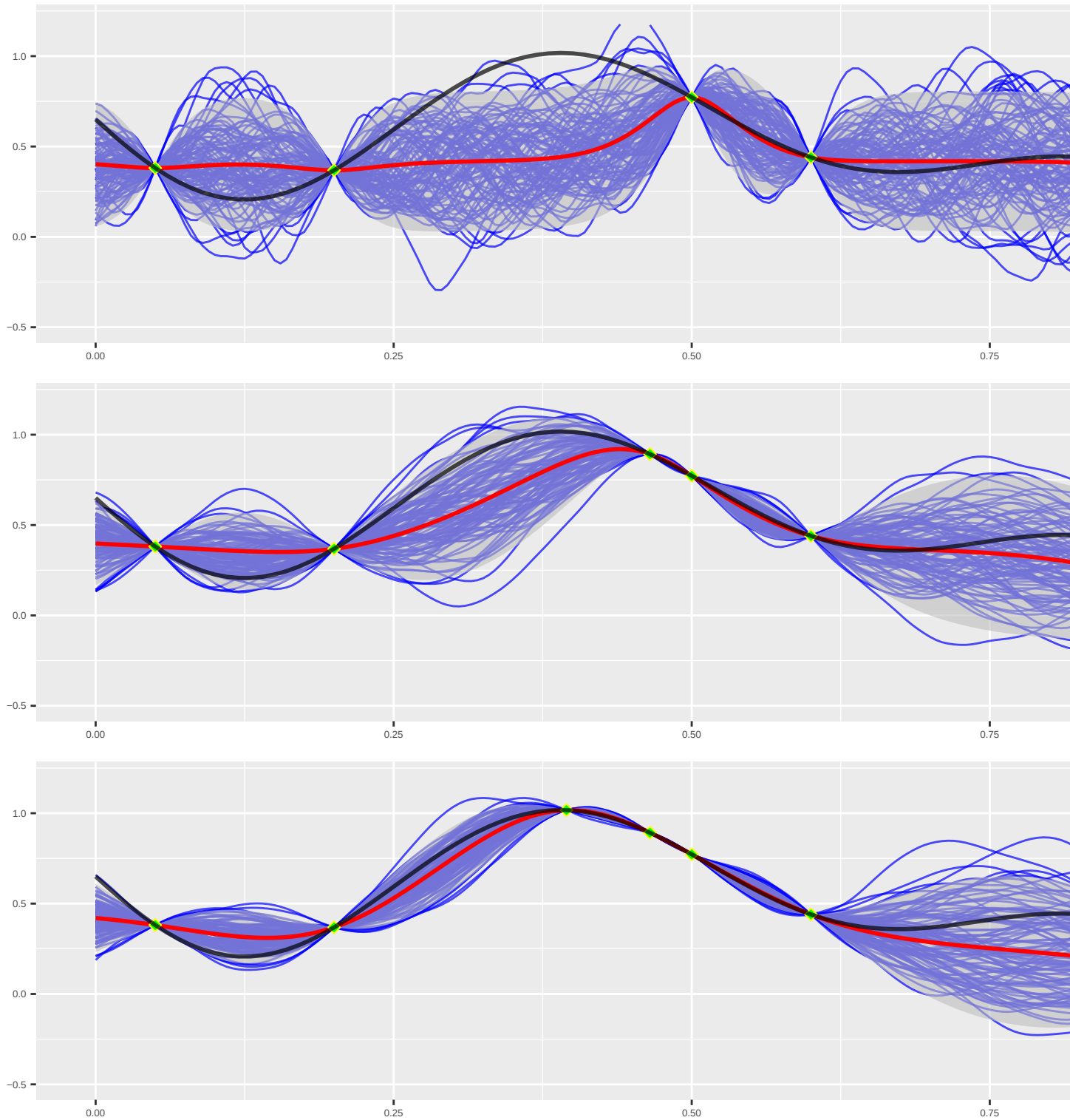
$$\mathcal{D} = \mathcal{D} \cup [x^{next}, f(x^{next})] = [[0.05, 0.2, 0.5, 0.6, 0.95, 0.46]; [0.38, 0.36, 0.77, 0.44, 0.16, 0.91]]$$

.

At this stage we perfomed first iteration of the BO. Now, the new observational data is used for modeling the probalitic model, then a new point is selected. This process sequentially, continues and it stops when we the solution satisfactory:

```
set.seed(123)
source("plot_func/plot_post.R")

# Making gp model and find the next x at iteration 1
x <- c(0.05,0.2,0.5,0.6,0.95)
obs_data <- df_x_y(x)
x_predict <- seq(0,1,0.005)

predict_list <- gp_model(obs_data,x_predict)
posterior_1 <- plot_post(predict_list$predict_list,x_predict,obs_data)

new_x_point1 <- utility_cal(predict_list$predict_list,x_predict,obs_data,0.1)
utility_1 <- utility_cal_plot(predict_list$predict_list,x_predict,obs_data,
                              0.1, new_x_point1)

##################

# Making gp model and find the next x at iteration 2

x2 <- c(0.05,0.2,0.5,0.6,0.95,new_x_point1)
obs_data2 <- df_x_y(x2)
x_predict <- seq(0,1,0.005)

predict_list2 <- gp_model(obs_data2,x_predict)
posterior_2 <- plot_post(predict_list2$predict_list,x_predict,obs_data2)
new_x_point2 <- utility_cal(predict_list2$predict_list,x_predict,obs_data2,0.1)
utility_2 <- utility_cal_plot(predict_list2$predict_list,x_predict,obs_data2,
                              0.1, new_x_point2)

###################

# Making gp model and find the next x at iteration 3

x3 <- c(0.05,0.2,0.5,0.6,0.95,new_x_point1,new_x_point2)
obs_data3 <- df_x_y(x3)
x_predict <- seq(0,1,0.005)

predict_list3 <- gp_model(obs_data3,x_predict)
posterior_3 <- plot_post(predict_list3$predict_list,x_predict,obs_data3)
```

```
new_x_point3 <- utility_cal(predict_list3$predict_list,x_predict,obs_data3,0.1)
utility_3 <- utility_cal_plot(predict_list3$predict_list,x_predict,
                              obs_data3,0.1, new_x_point3)

##################

library(gridExtra)
zz <- grid.arrange(posterior_1, utility_1, posterior_2, utility_2, posterior_3, utility_3,
                   ncol=2,
             widths = c(6, 4))
```

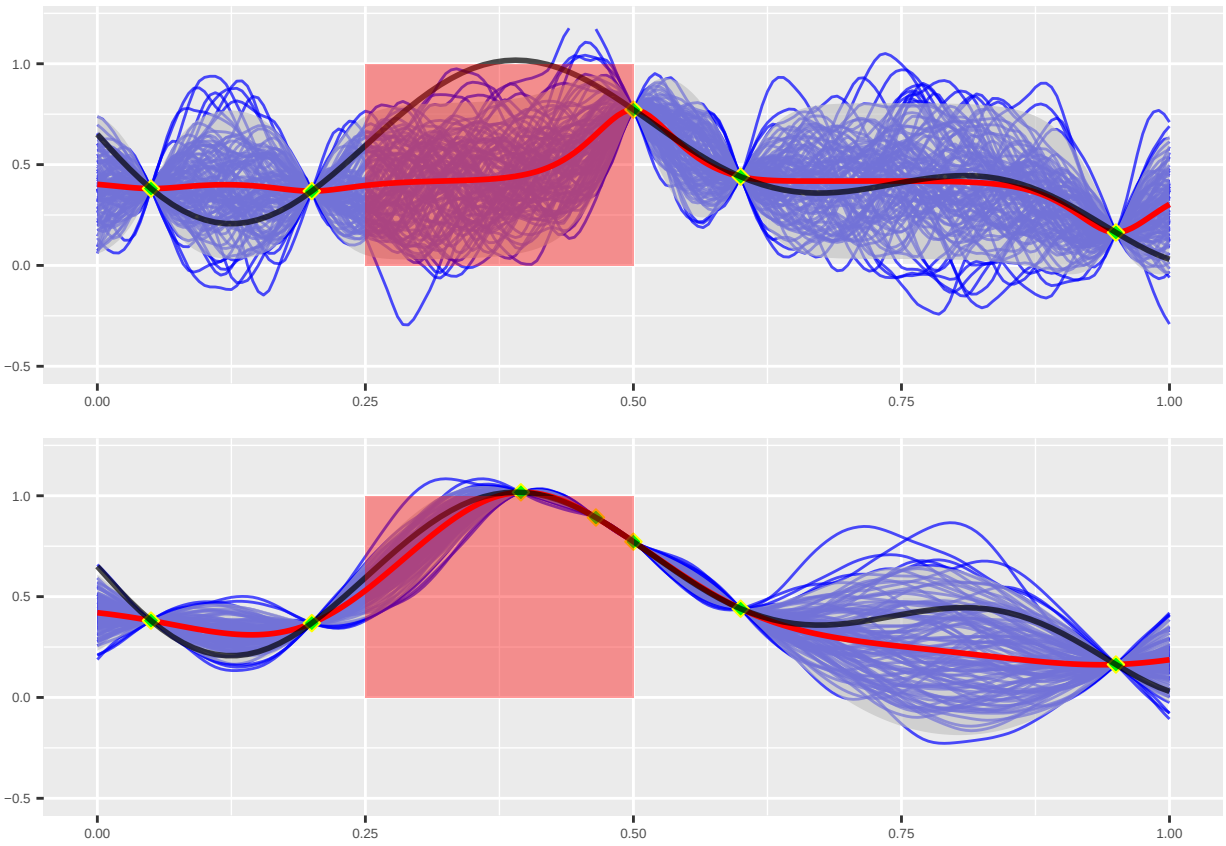



```
ggsave("output_plots/full_bo.png",zz,width = 20, height = 20, units = "cm", dpi=600)
```

Very nice that after only two iteration, we found the x=0.395, which is considering the optimal solution, x=0.39, with two digit we found optimal solution :)

```
init_pos <- posterior_1 +
  annotate("rect", xmin = 0.25, xmax = 0.5, ymin = 0, ymax = 1,
  alpha = .4, fill="red")

after_pos <- posterior_3 +
  annotate("rect", xmin = 0.25, xmax = 0.5, ymin = 0, ymax = 1,
  alpha = .4, fill="red")
```

```
compare_zone <- grid.arrange(init_pos, after_pos, ncol=1)
```




```
ggsave("output_plots/compare_bo.png",compare_zone,width = 20, height = 20, units = "cm", dpi=600)
```