Expected Improvement

Contents

```
import numpy as np
np.set_printoptions(precision=3)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(rc={"figure.dpi":100, "savefig.dpi":300})
sns.set_context("notebook")
sns.set_style("ticks")

import scipy.stats as st

def plot_1d_regression(
    x_star,
    gpm,
    ax=None,
    f_true=None,
    num_samples=10
):
    """Plot the posterior predictive.

    Arguments
    x_start  --  The test points on which to evaluate.
    gpm      --  The trained model.

    Keyword Arguments
    ax          --  An axes object to write on.
    f_true      --  The true function.
    num_samples --  The number of samples.
    """
    m_star, v_star = gpm.predict(x_star)

    f_lower = (
        m_star - 2.0 * np.sqrt(v_star - gpm.likelihood.variance)
    )
    f_upper = (
        m_star + 2.0 * np.sqrt(v_star - gpm.likelihood.variance)
    )

    if ax is None:
        fig, ax = plt.subplots()

    ax.plot(x_star, m_star, lw=2, label='$m_n(x)$')

    ax.fill_between(
        x_star.flatten(),
        f_lower.flatten(),
        f_upper.flatten(),
        alpha=0.5,
        label='$f(\mathbf{x}^*)$ 95% pred.'
    )

    if f_true is not None:
        ax.plot(
            x_star,
            f_true(x_star),
            'm-.',
            label='True function'
        )

    if num_samples > 0:
        f_post_samples = gpm.posterior_samples_f(
            x_star,
            num_samples
        )
        ax.plot(x_star, f_post_samples[:, 0, :], 'r', lw=0.5)
        ax.plot([], [], 'r', lw=0.5, label="Posterior samples")

    ax.plot(gpm.X,
            gpm.Y,
            'kx',
            markersize=5,
            markeredgewidth=2,
            label='Observations'
    )

    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$')

    plt.legend(loc='best')

    return m_star, v_star


def plot_iaf(
```

```
    x_star,

    gpr,
    alpha,
    alpha_params={},
    ax=None,
    f_true=None
):
    """Plot the information acquisition function.

    Arguments
    x_star       -- A set of points to plot on.
    gpr          -- A rained Gaussian process regression
                    object.
    alpha        -- The information acquisition function.
                    This assumed to be a function of the
                    posterior mean and standard deviation.

    Keyword Arguments
    ax           -- An axes object to plot on.
    f_true       -- The true function - if available.

    The evaluation of the informationa acquisition function
    is as follows:

        af_values = alpha(mu, sigma, y_max, **alpha_params)

    """
    if ax is None:
        fig, ax = plt.subplots()

    ax.set_title(
        ", ".join(
            f"{n}={k:.2f}"
            for n, k in alpha_params.items()
            )
    )

    m, v = plot_1d_regression(
        x_star,
        gpr,
        ax=ax,
        f_true=f_true,
        num_samples=0
    )

    sigma = np.sqrt(v)
    af_values = alpha(m, sigma, Y.max(), **alpha_params)
    next_id = np.argmax(af_values)
    next_x = x_star[next_id]
    af_max = af_values[next_id]

    ax2 = ax.twinx()
    ax2.plot(x_star, af_values, color=sns.color_palette()[1])
    ax2.set_ylabel(
        'Maximum Upper Interval',
        color=sns.color_palette()[1]
    )
    plt.setp(
        ax2.get_yticklabels(),
        color=sns.color_palette()[1]
    )
    ax2.plot(
        next_x * np.ones(100),
        np.linspace(0, af_max, 100),
        color=sns.color_palette()[1],
        linewidth=1
    )

def maximize(
    f,
    gpr,
    X_design,
    alpha,
    alpha_params={},
    max_it=10,
    optimize=False,
    plot=False,
    **kwargs
):
    """Optimize a function using a limited number of evaluations.

    Arguments
    f            -- The function to optimize.
    gpr          -- A Gaussian process model to use for representing
                    our state of knowldege.
    X_design     -- The set of candidate points for identifying the
```

```
                maximum.
alpha        -- The information acquisition function.
                This assumed to be a function of the
                posterior mean and standard deviation.

Keyword Arguments
alpha_params -- Extra parameters to the information
                acquisition function.
max_it       -- The maximum number of iterations.
optimize     -- Whether or not to optimize the hyper-parameters.
plot         -- Determines how often to plot. Make it one
                to plot at each iteration. Make it max_it
                to plot at the last iteration.

The rest of the keyword arguments are passed to plot_iaf().
"""
af_all = []
for count in range(max_it):
    # Predict
    m, sigma2 = gpr.predict(X_design)
    sigma = np.sqrt(sigma2)

    # Evaluate information acquisition function
    af_values = alpha(
        m,
        sigma,
        gpr.Y.max(),
        **alpha_params
    )

    # Find best point to include
    i = np.argmax(af_values)
    X = np.vstack([gpr.X, X_design[i:(i+1), :]])
    af_all.append(af_values[i])

    # Make observation
    y = np.vstack([gpr.Y, [f(X_design[i, :])]])
    # Update GPR
    gpr.set_XY(X, y)
    if optimize:
        gpr.optimize()

    # Plot if required
    if count % plot == 0:
        if "ax" in kwargs:
            ax = kwargs[ax]
        else:
            fig, ax = plt.subplots()
        plot_iaf(
            X_design,
            gpr,
            alpha,
            alpha_params=alpha_params,
            f_true=f,
            ax=ax
        )
        ax.set_title(
            f"N={count}, " + ax.get_title()
        )
return af_all
```

Objectives

- Develop intuition about the expected improvement

Expected Improvement

Let’s reintroduce the same running example as the previous hands-on activity.

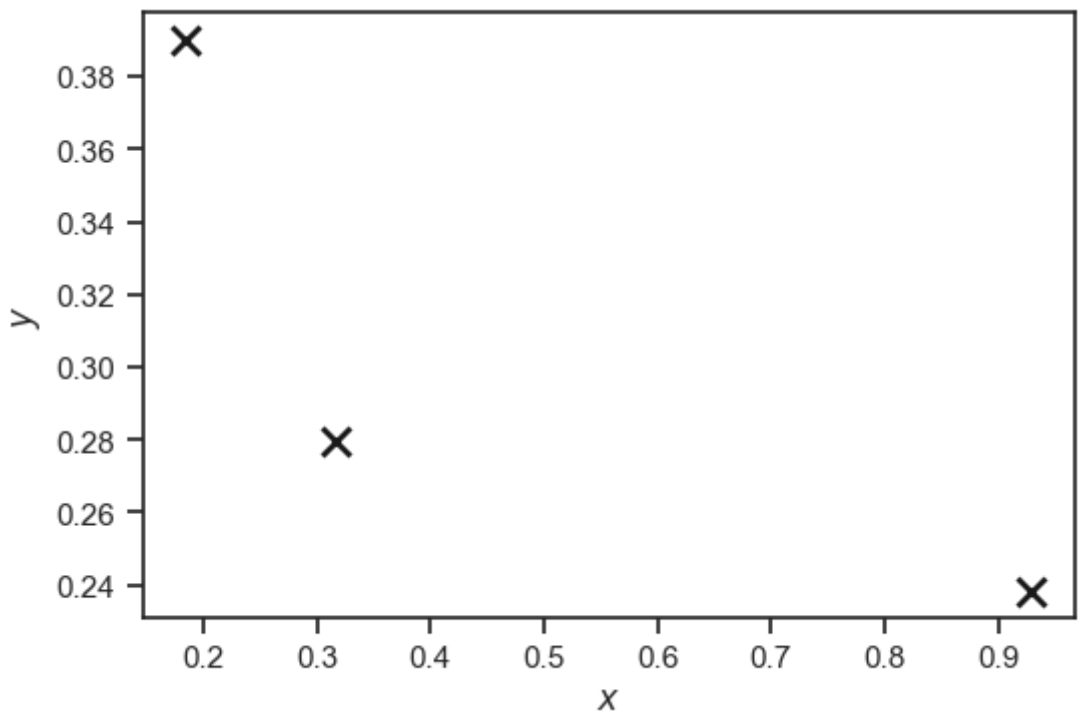
```
def f(x):
    """A function to optimize."""
    return 4 * (1. - np.sin(6 * x + 8 * np.exp(6 * x - 7.)))

np.random.seed(12345)

n_init = 3

X = np.random.rand(n_init)
Y = f(X)

plt.plot(X, Y, 'kx', markersize=10, markeredgewidth=2)
plt.xlabel('$x$')
plt.ylabel('$y$');
```



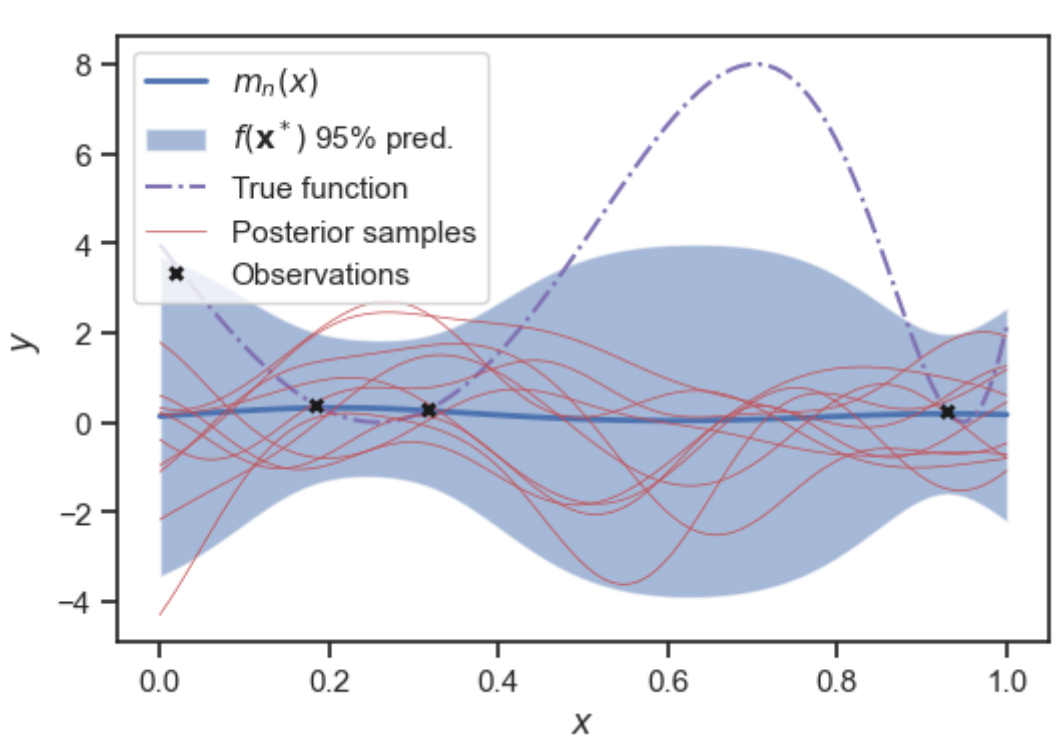
Just like in the previous hands-on activity, assume that we have made some observations and that we have used them to do Gaussian process regression resulting in the point-predictive distribution:

$$p(y|\mathbf{x}, \mathcal{D}_n) = \mathcal{N}\left(y|m_n(\mathbf{x}), \sigma_n^2(\mathbf{x})\right),$$

where $m_n(\mathbf{x})$ and $\sigma_n^2(\mathbf{x})$ are the predictive mean and variance respectively. Here is the code for this:

```
import GPy

k = GPy.kern.RBF(1, lengthscale=0.15, variance=4.)
gpr = GPy.models.GPRegression(X[:, None], Y[:, None], k)
x_star = np.linspace(0, 1, 100)[:, None]
plot_1d_regression(x_star, gpr, f_true=f);
```



The expected improvement is a bit more involved, but it serves as a template for deriving more general information acquisition functions. Here is how you think. Consider a hypothetical experiment at $\mathbf{x}$ and assume that you observed y. How much improvement is that compared to your currently best observed point y_n^*. It is:

$$I_n(\mathbf{x}, y) = \begin{cases} 0, & \text{if } y \leq y_n^*, \\ y - y_n^*, & \text{otherwise.} \end{cases}$$

But you don't know what y is. What do you do now? Well, the only legitimate thing to do is to take the expectation over what you expected y to be given what you know at that ~~moint~~ moment. So it is:

$$\begin{aligned}\mathrm{EI}_n(\mathbf{x}) &= \int_{-\infty}^{\infty} I_n(\mathbf{x}, y) p(y|\mathbf{x}, \mathcal{D}_n) dy \\ &= \int_{-\infty}^{y_n^*} 0 \cdot p(y|\mathbf{x}, \mathcal{D}_n) dy + \int_{y_n^*}^{\infty} (y - y_n^*) \cdot p(y|\mathbf{x}, \mathcal{D}_n) dy.\end{aligned}$$

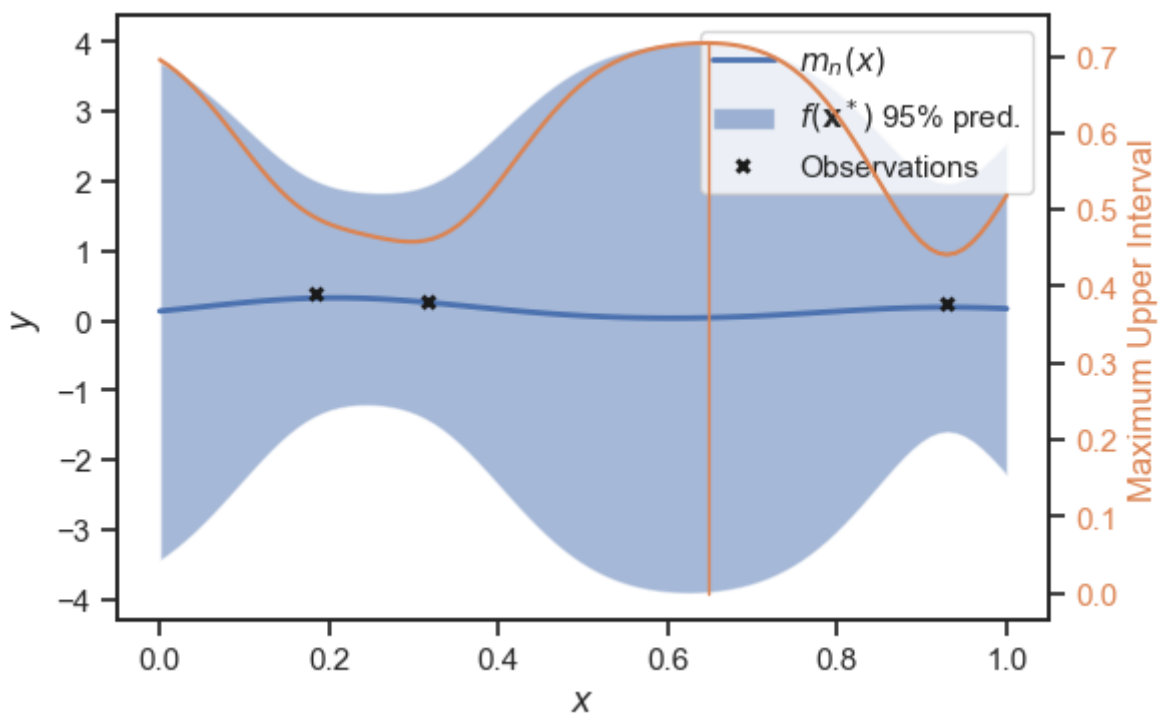
You can work this out analytically. You will get:

$$\mathrm{EI}_n(\mathbf{x}) = \frac{m_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})} \Phi\left(\frac{m_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right) + \phi\left(\frac{m_n(\mathbf{x}) - y_n^*}{\sigma_n(\mathbf{x})}\right).$$

```
def ei(m, sigma, ymax):
    """Return the expected improvement.

    Arguments
    m     -- The predictive mean at the test points.
    sigma -- The predictive standard deviation at
             the test points.
    ymax  -- The maximum observed value (so far).
    """
    u = (m - ymax) / sigma
    ei = sigma * (u * st.norm.cdf(u) + st.norm.pdf(u))
    ei[sigma <= 0.] = 0.
    return ei

plot_iaf(
    x_star,
    gpr,
    ei
)
```



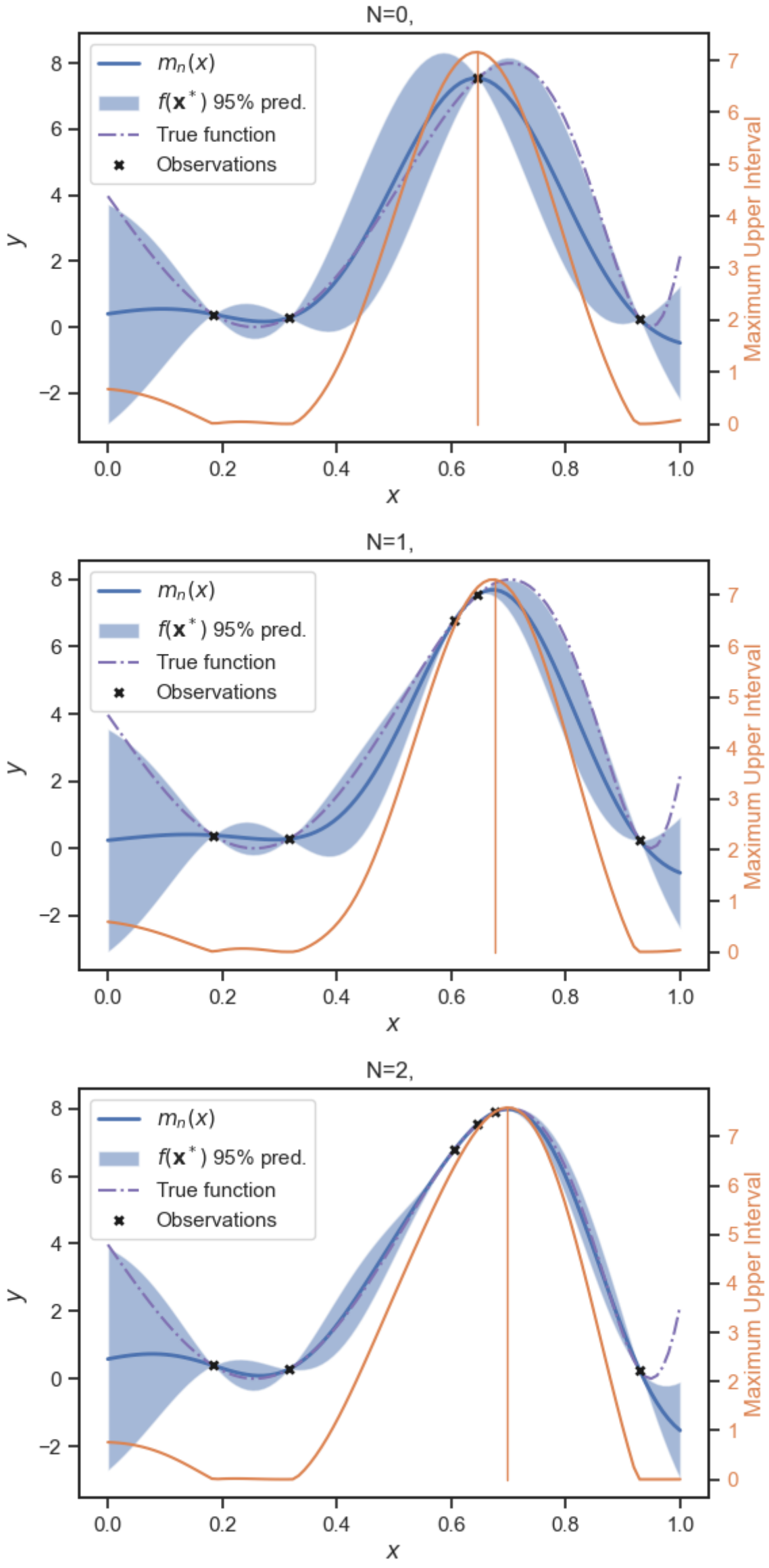
Bayesian global optimization with the expected improvement

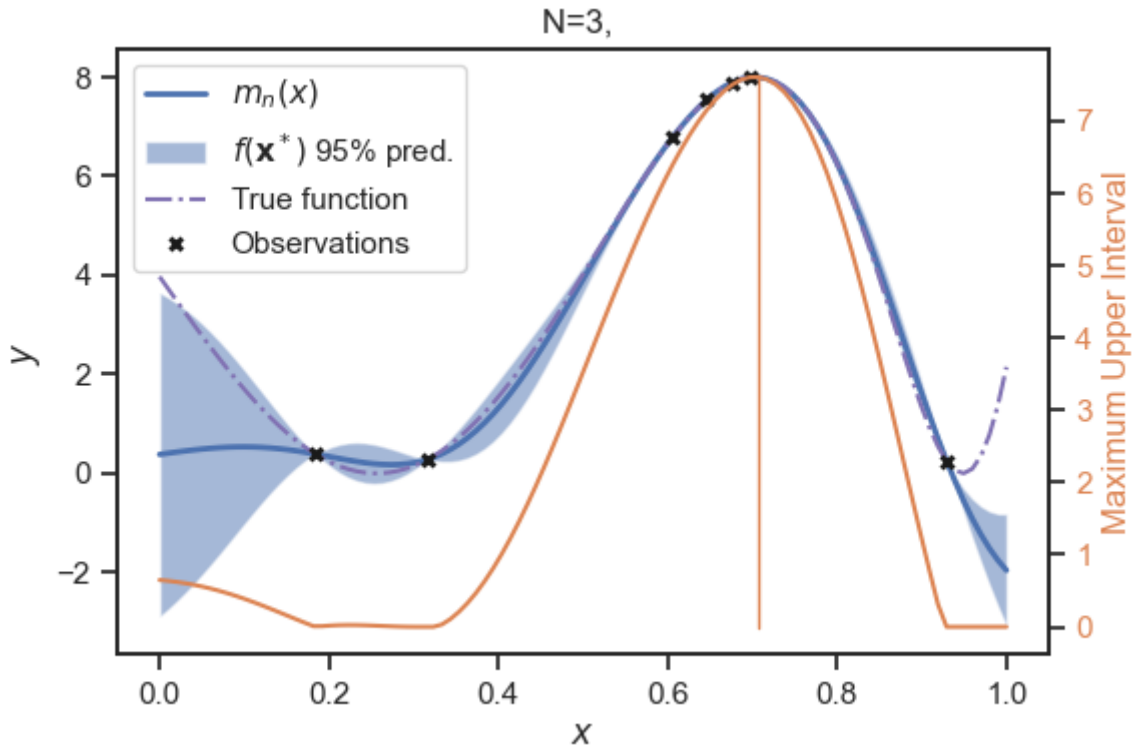
Let's now run the Bayesian global optimization algorithm using the expected improvement as the information acquisition function:

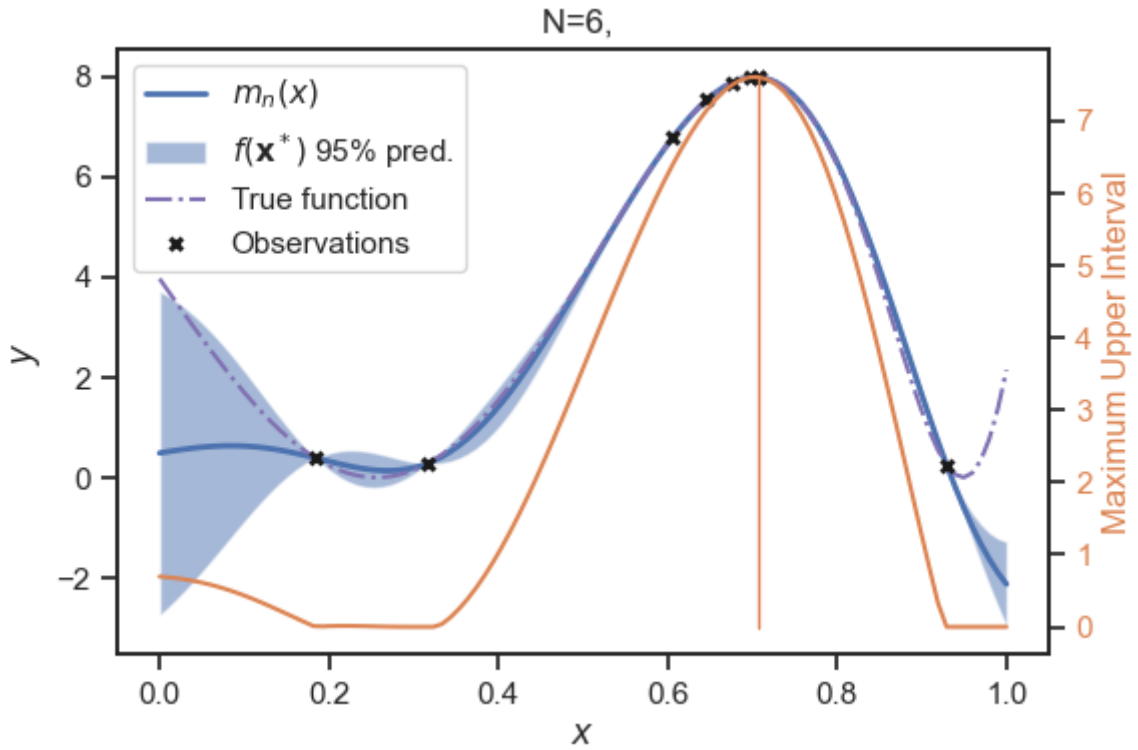
And here is how we can use the code with the probability of improvement:

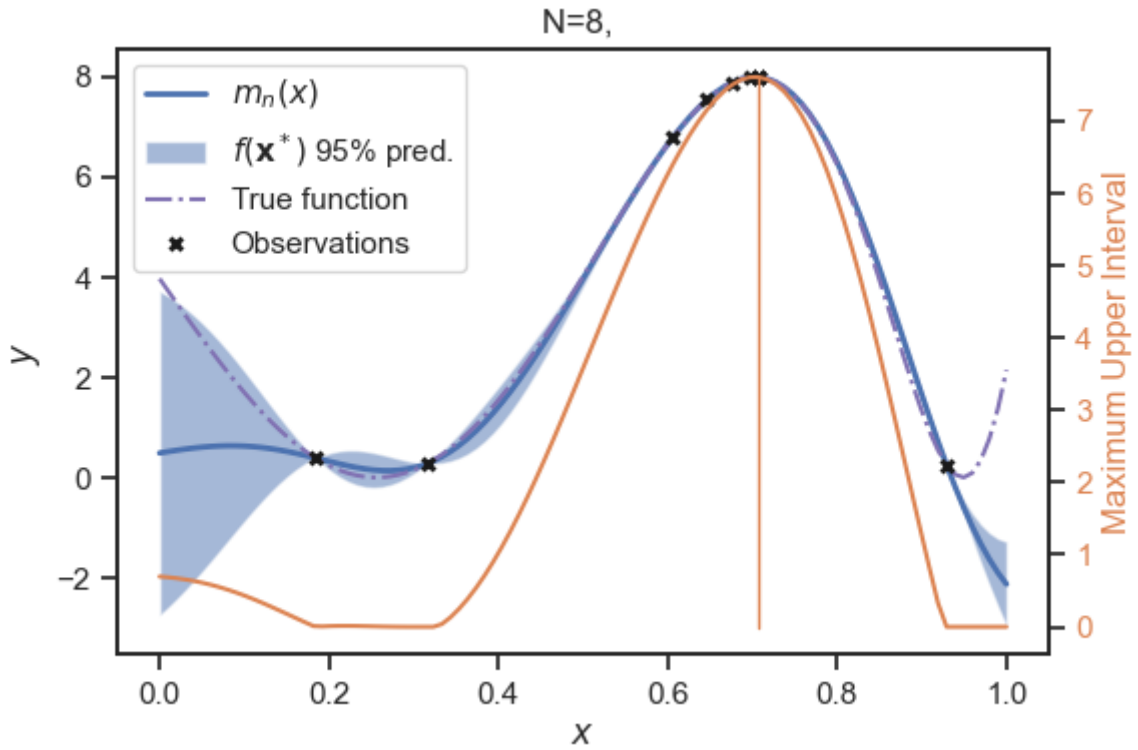
```
# Prepare the initial statistical model
k = GPy.kern.RBF(1, lengthscale=0.15, variance=4.)
gpr = GPy.models.GPRegression(X[:, None], Y[:, None], k)
gpr.likelihood.variance.constrain_fixed(1e-16)

# Run the algorithm
af_all = maximize(
    f,
    gpr,
    x_star,
    ei,
    max_it=10,
    plot=1
)
```


N=0,
N=1,
N=2,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval
0.0
0.2
0.4
0.6
0.8
1.0
−2
0
2
4
6
8
0
1
2
3
4
5
6
7


N=3,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval
N=4,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval
N=5,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval


N=6,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval
N=7,
$m_n(x)$
$f(\mathbf{x}^*)$ 95% pred.
True function
Observations
y
x
Maximum Upper Interval



Questions

- Rerun the main algorithm for EI by optimizing the hyper-parameters. Hint: Go through the code of `maximize`.

By Ilias Bilionis (ibilion[at]purdue.edu)