Machine Learning in the Physical World

Sequential Decision Making Under Uncertainty

Carl Henrik Ek

October 31, 2024

Abstract

This week we are going to make use of our previous knowledge in modelling and use it for black-box optimisation. The idea here is that we have a function that we want to optimise, i.e. find the minima[1], the tricky thing is that we do not know the form of the function but we can evaluate it. This means that we will use a Gaussian process to specify a *surrogate* for the unknown function and use our belief in this surrogate as a guide to our search. When we build statistical models we quantify our knowlege using probabilities which specifies our beliefs. This means that we specify not only *what we know* but also *how well we know* it. The main thing that we want to get across in this lab is showing how useful and important this concept can be.

Lets assume that we have a function $f(\mathbf{x})$ that is *explicitly* unknown that we want to find the minima of. We will further assume that it is possible to evaluate the function but that each evaluation is expensive. This means that the problem that we have on our hands is to *search* the input domain for the extreme point but do so in a manner that we minimise the number of evaluation that we make of the function. One approach to address this type of problem is to use a technique called *Bayesian Optimisation* and that is the focus of this lab.

Searching for the extremum of explicitly unknown functions is problem that appears in many applications. The first use of Bayesian methods for approaching this is usually attributed to Močkus, 1975 a Lithuanian mathematician. This important work was slightly overlooked at the time but recently it has gotten the attention that it deserves. The reason for this is that with increasingly complicated models with enormous cost for training being able to efficiently utilise the data have become very important. The use of Bayesian optimisation for learning how to set parameters in complicated unstructured models[2] is often attributed to Snoek et al., 2012 who really put these types of techniques at the forefront of modern machine learning. Even though the are used everywhere this is often not reported particularly well, as an example it took several years for the authors of AlphaGo to properly publish and discuss the importance of Bayesian optimisation for their task Chen et al., 2018.

The main part of a Bayesian optimisation system is a loop where we in an iterative manner decided on new locations to evaluate the objective function. The two components of the loop are a surrogate model of the function which describes how we believe the function looks in every part of the domain and a acquisition function which decides based on our current belief of what the function is where to sample next. Importantly this makes it key to have uncertainty in our system as we need to have a belief about what the function value is everywhere. As we have already looked at Gaussian processes as a rich class of function priors we will use them for our surrogate model. Lets begin by making the problem more concrete.

Lets assume that we want to find the minima of a function $f(x)$,

$$\hat{x} = \text{argmin}_x f(x), \tag{1}$$

[1] we will usually refer to it as the minima, when we want to maximise, we just negate the objective function

[2] read neural networks

we will at each time have observed a set of values of the function at specific function locations, we will refer to this as the data $\mathcal{D} = \{\mathbf{x}, \mathbf{f}\}$. Furthermore at any point we have a current best estimate, we will refer to this location in the data space as $\hat{x}$ and its function value as $\hat{f} = f(\hat{x})$. We will use a surrogate model to describe our beliefs about the function f. We will use a Gaussian process to do so which means that we have access to a distribution $p(f|x, \mathcal{D})$ which is the predictive posterior of the Gaussian process.

1 Surrogate Model

We will use a Gaussian process as a surrogate model for the function. Last week we looked at how to work with Gaussian processes. If you feel that you need a bit of a recap of this go back to that lab and make sure that you understand Eq. 10-12 and how Figure 2 was generated. What we need to know from our GP is given a set of data $\mathcal{D}$ what is our belief of what the function is at every other location in the input domain. This is the object that we call the *predictive posterior* of the Gaussian process,

$$p(\mathbf{f}_*|\mathcal{D}, \mathbf{x}_*, \theta) = \mathcal{N}(\mu_{\mathbf{x}_*|\mathbf{x}}, K_{\mathbf{x}_*|\mathbf{x}}) \tag{2}$$

$$\mu_{\mathbf{x}_*|\mathbf{x}} = k(\mathbf{x}_*, \mathbf{x})k(\mathbf{x}, \mathbf{x})^{-1}\mathbf{f} \tag{3}$$

$$K_{\mathbf{x}_*|\mathbf{x}} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{x})k(\mathbf{x}, \mathbf{x})^{-1}k(\mathbf{x}, \mathbf{x}_*). \tag{4}$$

You will need to implement a function that can return the mean and the variance at a set of locations `x_star` of a Gaussian process parametrised using `theta`.

Code

```
def surrogate_belief(x,f,x_star,theta):

    return mu_star, varSigma_star
```

Now when we have our surrogate model set up it is time to move on to the second component, the acquisition function.

2 Aquisition Function

The idea of the aqusition function is that it encodes the strategy of how we should utilise the knowledge that we currently have in order to decide on where to query the function. The design of this function is where we balance the two important factors, *exploration* where we learn about new things, and *exploitation* where we utilise what we currently know. There are many different acquisition functions to use and we will here only look at one of them but in principle they all describe a *utility-value* across the whole input domain of how much we will "gain" by querying the function in this specific place.

2.1 Expected Improvement

The most commonly used acquisition function is *Expected Improvement* Močkus, 1975. The idea underlying expected improvement is that the utility of a location in the input domain is relative to how much lower we expect the function value at this point to be compared to the current best estimate. This means that the utility function $u(x)$ can be defined as follows,

$$u(x) = \max(0, f(x_*) - f(x)) \tag{5}$$

This means that we have a reward for every location in the space where the function $f(x)$ is smaller than the current best estimate $f(x_*)$. Now as we do not know $f(x)$ we want to use our knowledge from the surrogate model f. This we can do by taking the expectation of the utility function over our belief in the function as,

$$\alpha(x) = \mathbb{E}\left[u(x)|x, \mathcal{D}\right] = \int_{-\infty}^{f(x_*)} (f(x_*) - f(x))\mathcal{N}(f|\mu(x), k(x, x))\mathrm{d}f. \tag{6}$$

Note how the upper limit of the integral is the current best estimate of the function thereby implementing the max operator in Eq. 5.

One of the nice things about Expected improvement is that we can evaluate the expectation in closed form resulting in the following acquisition function,

$$\alpha(x) = \underbrace{(f(x_*) - \mu(x))\Psi(f(x_*)|\mu(x), k(x,x))}_{\text{exploitation}} + \underbrace{k(x,x)\mathcal{N}(f(x_*)|\mu(x), k(x,x))}_{\text{exploration}} \tag{7}$$

$$\Psi(f(x_*)|\mu(x), k(x,x)) = \int_{-\infty}^{f(x_*)} \mathcal{N}(f|\mu(x), k(x,x))\mathrm{d}f. \tag{8}$$

The function Ψ is the *cumulative density function* or *cdf* of the Gaussian which has the following form,

$$\Psi(x \mid \mu, \sigma) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right), \tag{9}$$

where $\mathrm{erf}(\cdot)$ is the *error-function*[3]. You do not have to implement this yourself as it is available as `scipy.stats.norm.cdf`. Now we want to choose points in the input domain that will maximise the acquisition function. Looking at the function that we have derived we can see that it includes two terms, the first term can be increased by picking an x value such that the difference between $f(x_*) - \mu(x)$ is large. In effect this is *exploiting* the knowledge that we currently have about the function. The second term can be increased by finding a location in the input domain such that $k(x,x)$ is large, i.e. the variance at this location is high. In effect this is *exploration* as we are looking for locations where we are uncertain of what the value is. As you can see these two terms formulates a specific balancing between the two key aspects of search, *exploration* and *exploitation*.

Now we need to write an implementation of the acquisition function we are going to need something looking like this,

Code

```
from scipy.stats import norm
    def expected_improvement(f_max, mu, varSigma, x):

        # norm.cdf(x, loc, scale) evaluates the cdf of the normal distribution

        return alpha
```

where `mu` and `varSigma` is the mean and the variance of the predictive posterior of the surrogate model at locations `x` which is the set of candidates for where to pick the next function evaluation from.

We now have all the parts that we need in order to implement our Bayesian optimisation loop, the surrogate model using a Gaussian process and the acquisition function using expected improvement.

3 Experiments

We will now write up the Bayesian optimisation loop that we will iterate through. The first thing we need is a function to evaluate. As we want to be able to play around with the function a bit we will add a set of possible arguments. The functions is the classical Forrester function that was proposed in Forrester, 2008. Initially we will drop the linear and constant term but you can alter them to test the performance. For the experiment we will limit the domain to $x \in [-1, 2]$.

[3] `https://en.wikipedia.org/wiki/Error_function`

Code

```
def f(X, noise=0.0):
    return -(-np.sin(3*X) - X**2 + 0.7*X + noise*np.random.randn(*X.shape))
```

The next thing that we will do is to decide on a finite set of possible evaluations of the function. The function that we are using is a function in $\mathbb{R}$ what we will do is to divide up this space into a finite set of locations and then our aim is to find at which one of these points we have the minimal value of the function. If we call this set $\mathbf{X}$ we will now start our loop by taking a random set of starting points, compute the predictive posterior over the remaining points, compute the acquisition for all the points not included in the model, pick the location with the highest acquisition and include this into the modelling set. A hand-wavy structure should look something like this.

Algorithm 1 Bayesian Optimisation

procedure BO$(f(x), \alpha(x), \mathbf{X})$
  $x_{\text{start}} \subset \mathbf{X}$ ▷ Pick a random set of start-points
  $x \leftarrow x_{\text{start}}$
  $f' \leftarrow \text{argmin}_{x' \in x} f(x')$
  while iter **do** ▷ Loop until we have reached max number of iterations
    Evaluate $\mu_{\mathbf{X}|\mathbf{x}}$ and $K_{\mathbf{X}|\mathbf{x}}$ ▷ Predictive Posterior of Surrogate
    Evaluate $\alpha(\mathbf{X})$ ▷ Aquisition Function
    $x' = \text{argmax}_{\hat{x} \in \mathbf{X}} \alpha(\hat{x})$ ▷ Pick "best" candidate to evaluation set
    $x = x \cup x'$ ▷ Add element x' to the set
    if $f(x') < f'$ **then** ▷ Update current minima
      $f' = f(x')$
    end if
  end while
  return f'
end procedure

One useful way to code this things is to keep two sets of points, you first start with an array with all locations that you can evaluate the function at, then you pick a random subset from this and move them to another set. Then for each evaluation you keep removing points from the initial set. This can easily be done with numpy arrays like this,

Code

```
# remove points from an array
   x_2 = np.arange(10)
   index = np.random.permutation(10)
   x_1 = x_2[index[0:3]]
   x_2 = np.delete(x_2, index[0:3])

   # remove largest element
   ind = np.argmax(x_2)
   x_1 = np.append(x_1, x_2[ind])
   x_2 = np.delete(x_2, ind)
```

When you got the loop implemented you can try and see how good result you generally get in a fixed number of iterations. Then you compare this result with taking the same number of locations uniformly at random from the index set and evaluating them. If you compare the runs how often do you get a better value with the Bayesian optimisation approach compared to the random search? Now we can alter this question slightly,

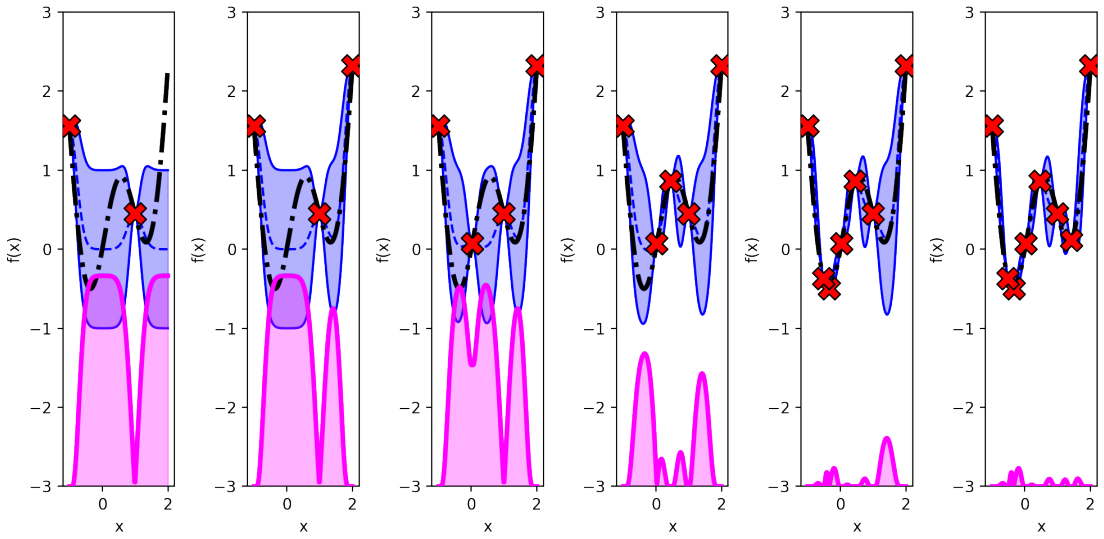

Figure 1: *The image below shows the aqcuisition function in magenta, the true function in black and the surrogate models belief in blue. The left-most pane is the first iteration, at each iteration we add in the location of the highest acuisition and update the surrogate model. The image on the far right shows the 8th iteration. The plot was generated with a zero-mean GP with a squared exponential co-variance function with* `lengthscale=0.1`, `variance=2.0` *and a* `noise=0.1`.

given that you have a current best estimate using BO, how many random samples do you need in order to get an equally good result?

4 Experiments II

Now we have implemented a simple BO example it is time to move to something slightly more interesting. The idea that we wanted to get across was that beliefs matter when you are reasoning about the unknown and I hope this has come across. Now if we take a few steps back and think about the beginning of the module we talked about Laplace Demon and one of its important messages was that *you can only ever reason about data in light of your beliefs*. Another way of saying this is that dependent on what you believe you will interpret data differently. It is amusing to think of conspiracy theories here[4] where a believer will interpret everything to be supporting their theory simply because they have such a strong belief[5]. Now if we come back to our simple optimisation example in Figure 1 if you do not know the true function can you ever be sure that you have found the extremum? No you cannot because the function is explicitly unknown. If this is the case how do we know when to stop? Again this comes back to your belief about the function. By choosing the prior we did we explicitly encoded our assumption about the function. In the experiment we ran we use a exponentiated quadratic covariance where we encoded a belief in the smoothness of the function. Or in other words how rapidly the function changes. This is the key concept in this lab, *your posterior belief can only ever be interpreted in "light" of your prior belief.*

4.1 GPyOpt

In the previous example we implemented the BO loop from scratch, I hope this allowed you to see how simple the basic mechanics, and how intuitive the sequential decision making loop is. However, there was one thing we did not do and that was to update our belief of the function with data. What we will now add

[4] and Laplace does so when discussing Horoscopes in his book

[5] if you have ever tried to reason with someone who believes in a flat earth you will know what I mean.

is an additional loop meaning that we at each iteration try to estimate the hyper-parameters that govern the function model. Now this might feel like an odd thing to do as we become very sensitive to the data that we actually gather which might lead to a significant model mismatch between the actual function and the model we have. Several authors have shown this in practice and there has been recent work that aims at reformulating the role of the function prior for example Bodin et al., 2020. In the traditional BO loop this is still what you do, at each iteration you estimate your hyper-parameters from the data that you currently have access to. This makes the loop quite expensive to run which means we would do well to have a bit more efficient implementation. We will therefore from now on use a software package called `GPyOpt` which is completely open-source and very mature. You can install both packages using `pip`.

To begin with we will first try a similar experiment to that one we tried before. We will do a simple function with a couple of parameters that you can alter if you want to test different behaviours of the functions.

Code

```
import numpy as np

def f(x, beta=0.2, alpha1=1.0, alpha2=1.0):
    return np.sin(3.0*x) - alpha1*x + alpha2*x**2 + beta*np.random.randn(x.shape[0])
```

Rather than defining the GP directly this will be done internally by `GPyOpt` using the package `GPy`. What is left for us is to specify the prior assumption we have over the function, i.e. specify the co-variance function of the GP.

Code

```
import GPy
kernel = GPy.kern.RBF(input_dim=1, variance=1.0, lengthscale=4.0)
```

Now the last thing that remains is to provide `GPyOpt` with the parameters and structure of the problem. This is done through a `dictionary` which we will here call domain. It specifies the variables that should be optimised, the type of the variable and the domain of the problem.

Code

```
domain = [{'name': 'var_1', 'type': 'continuous', 'domain': (-3,3)}]
```

Now the problem is specified, we have defined our prior assumptions about the function and we can let `GPyOpt` take care of the loop for us and give us back the result.

Code

```
from GPyOpt.methods import BayesianOptimization

opt = BayesianOptimization(f=f, domain=domain,model_type='GP',
                           initial_design_numdata = 1,
                           kernel=kernel, acquisition_type='EI')
opt.run_optimization(max_iter=6)
opt.plot_acquisition()
```

Run the code above, play around with more challenging functions so that you get a feeling for how the software works. The documentation of both `GPy` and `GPyOpt` is excellent and should be able to answer most of your queries around how things work.

4.1.1 Prior Knowledge

Now you can only every formulate very generic priors if you do not know what you are modelling so let us take a more specific example that will allow us to formulate a more informative prior. Let us take the temperature variations over a year on a planet with 10 days in a year. Now we know quite a bit about this system and this is something that we should exploit in our prior. First, temperatures can be expected to be periodic on several different scales, there is a daily cycle and there is also a yearly cycle. Secondly we know that temperatures are rising so there is a probably a trend that it is likely to get warmer every day. Lets say that we have a temperature model as follows, where we know the underlying structure but we are uncertain of the actual impact of each of the of the factors.

$$f(x) = \alpha sin(\frac{x}{10}2\pi) + \beta sin(\frac{x}{0.5}2 * \pi) + \gamma x \tag{10}$$

We will also assume that the noise in the measurements is a zero mean Gaussian with variance 0.1 such that,

$$y_i = f(x_i) + \epsilon \tag{11}$$

$$\epsilon \sim \mathcal{N}(0, 0.1^2) \tag{12}$$

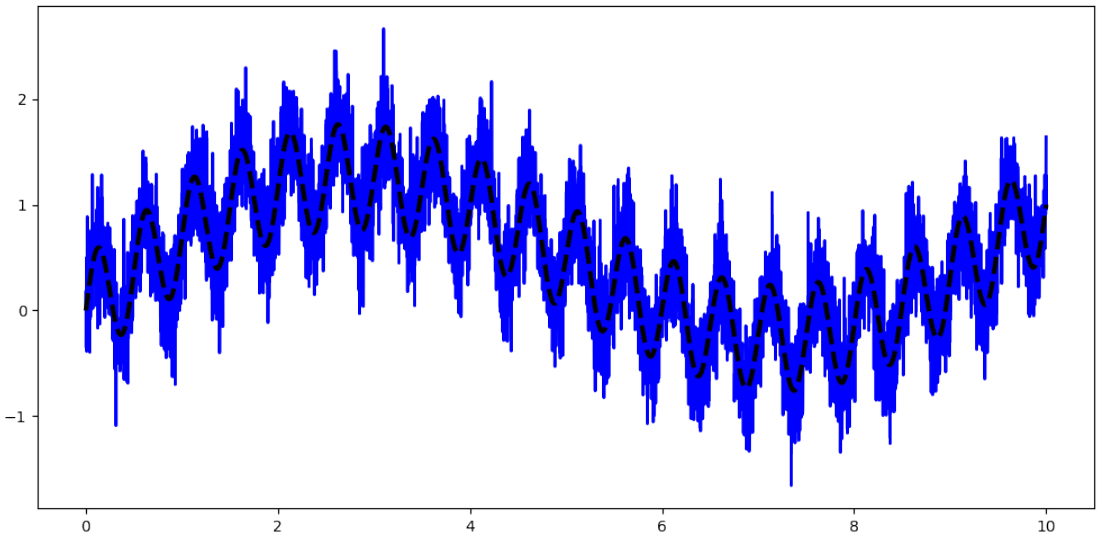

Figure 2: *The image shows the true function in* `black` *and the noisy measurements in* `blue` *generated with the following parameters* $\{\alpha, \beta, \gamma\} = \{1.0, 0.5, 0.2\}$.

Our task is now to find the lowest temperature of the year. First try and run this experiment with the same co-variance function as we used before, only encoding the range of values we expect using the variance of co-variance function and the degree of smoothness using the length-scale the resulting plot is shown in Figure 3. As you can see this performs very poorly and you recover very little of the structure of the function and its quite unlikely[6] that you will find the minima which is sometime during day 7. To proceed we can now introduce the knowledge that we have of the system and use the fact that we know that the function is an additive composition of four terms. This is really simple to within `GPy` as we can specify what is referred to as a compound co-variance by adding co-variance functions together. This can be seen in the code below.

[6] the procedure is stochastic so you might get lucky

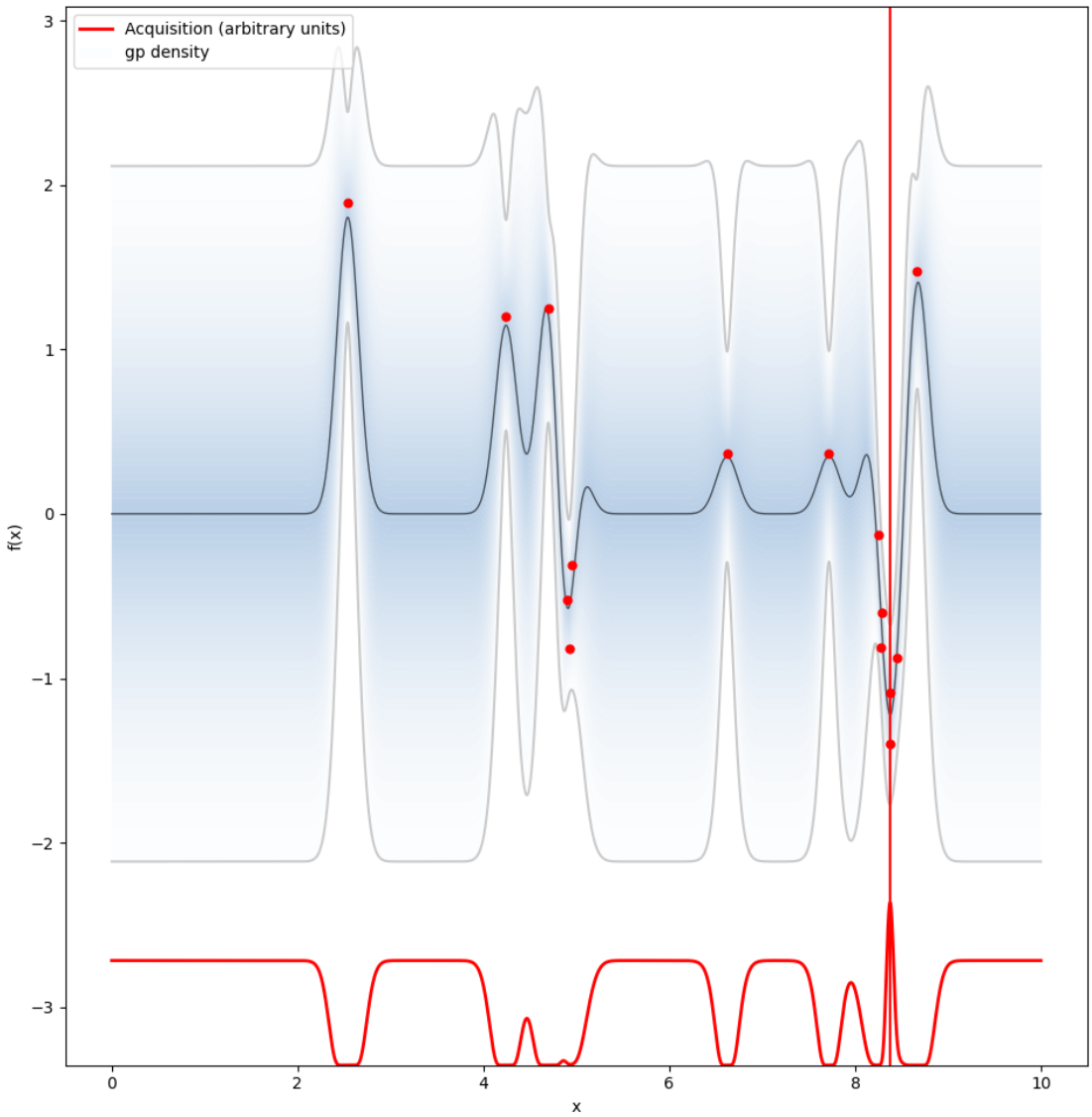

Figure 3: *The plot above shows the BO loop applied to the temperature data where we have not included the knowledge that we have about the system.*

Code

```
import GPy
from GPyOpt.methods import BayesianOptimization
import numpy as np

def f(x, alpha=1.0, beta=0.5, gamma=0.2):
    return alpha*np.sin(2*np.pi*x/10) + ((beta*np.sin((2*np.pi*x/0.5))) +
                                         (gamma*x) +
                                         (0.1*np.random.randn(x.shape[0])))

kernel_rbf = GPy.kern.RBF(input_dim=1, variance=1.0, lengthscale=4.0)
kernel_cmpnd = ((GPy.kern.RBF(input_dim=1, variance=1.0, lengthscale=4.0)) +
                (GPy.kern.StdPeriodic(input_dim=1, variance=1.0, period=10.0)) +
                (GPy.kern.StdPeriodic(input_dim=1, variance=1.0, period=0.5)) +
                (GPy.kern.Linear(input_dim = 1)) +
                (GPy.kern.White(input_dim=1, variance=0.1)))
domain = [{'name': 'var_1', 'type': 'continuous', 'domain': (0,10)}]

opt = BayesianOptimization(f=f, domain=domain,model_type='GP',
                           kernel=kernel_cmpnd,
                           acquisition_type='EI',
                           initial_design_numdata=5)
opt.run_optimization(max_iter=10)
opt.plot_acquisition()
```

In addition to altering the co-variance function we have also made an additional change namely increased the `initial_design_numdata` to 5. This parameter specifies how many random points that will be used before the acquisition function gets evaluated. Using a very small number means that we run the risk of being too reliant on the initial values. In Figure 4 the result of including the structure is shown. As you can see we now have a much better knowledge of the underlying signal due to us structuring the hypothesis space in a manner much more in line with the generating process of the data. Even though we have not found the correct day after 10 iterations we are much closer and will within a small number of iterations likely reach this.

5 Summary

Hopefully you have seen that having the concept of uncertainty can be really useful in order to direct a sequential search strategy as in Bayesian optimisation. Hopefully the simple one dimensional example provided you with an intuition and I hope that it feels like quite a natural way of how to search for an optima and that you can relate to the balancing between exploration and explotation in how you would most likely approach a problem. With the second example we wanted to show the value of including the knowledge that we have. Importantly the prior that we specify is still a Gaussian process which means that we have not removed any smooth functions from our distribution, everything is still possible. However, what we have done is that we have restructured where the prior places mass and included that our belief about functions with a specific periodic structure is significantly higher compared to other functions. So again, this is evidence for the important of beliefs and when working with real physical systems we often have a significant amount of knowledge and this is what we should aim to include.

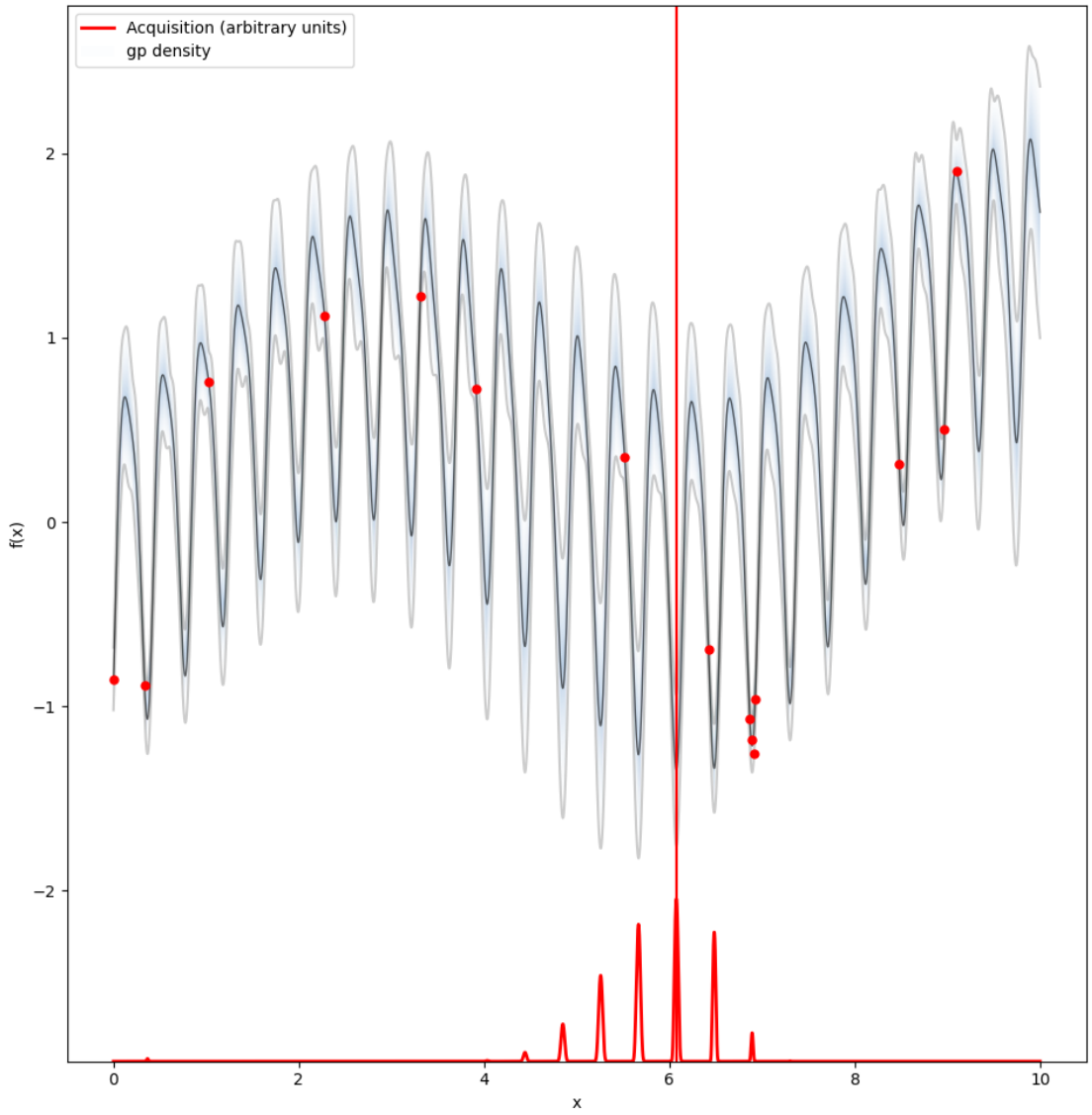

Figure 4: *The plot above shows the BO loop applied to the temperature data where we have included the knowledge that we have about the system. It is clear that including the knowledge of the periodicity of the system allows for a much better balance between exploration and explotation.*

References

Bodin, Erik et al. (2020). "Modulating Surrogates for Bayesian Optimization." In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2019, 12-18 July 2020, Virtual*.

Chen, Yutian et al. (2018). "Bayesian Optimization in Alphago." In: *CoRR*. arXiv: 1812.06855 [cs.LG].

Forrester, Alexander (2008). *Engineering design via surrogate modelling : a practical guide*. Chichester, West Sussex, England Hoboken, NJ: J. Wiley. ISBN: 9780470060681.

Močkus, J. (1975). "On bayesian methods for seeking the extremum." In: *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*. Ed. by G. I. Marchuk. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 400–404. ISBN: 978-3-540-37497-8.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 2951–2959.