Department of Statistics and Actuarial Science
University of Hong Kong

Tutorial 12: Connections of the Algorithms

ZHANG Yan

November 28, 2021

Overview

MCMC and Gradient Method

MCMC and Gradient Method

Problem formulation

- Generally, there are two ways to solve a problem: 1). **Sampling** from $\pi(x)$, where $\pi(x) \geq 0$ is integrable; 2). Do **optimization** for $f(x)$, where we assume there is a global maximum.
- The sampling is about **exploration**, which means we want to better know the sampling target $\pi(x)$ itself; The optimization is about **exploitation**, which means we simply want to exploit the optimization target $f(x)$ to obtain the optimal point.
- **MCMC** and the **gradient method** (gradient ascent or gradient descent) are the most popular algorithms in their own fields respectively. They are like counterparts to each other.
- MCMC requires no additional conditions; The gradient method only needs the existence of the first derivative.
- There are also **optimization-based sampling** (Adaptive Importance Sampling in hw1; **Hamiltonian Monte Carlo**) and **sampling-based optimization** (Annealing; **Thompson Sampling**). (Thompson Sampling will be introduced in Bayesian Optimization part later.)

- **Hamiltonian Monte Carlo** (**HMC**), or called Hybrid Monte Carlo, is a combination of the two popular methods, MCMC and the gradient method.
- The original **Random Walk**:

$$y = x_n + \varepsilon u,\ u \sim N(0,1),$$

let $x_{n+1} = y$ with the acceptance probability $\alpha = \min(1, \pi(y)/\pi(x_n))$.

- The original **gradient descent**:

$$x_{n+1} = x_n - \varepsilon \frac{\partial h}{\partial x}(x_n),\ h(x) = -\log(\pi(x)).$$

- The naive hybrid algorithm:

$$x_{n+1} = x_n - \frac{\varepsilon^2}{2}\frac{\partial h}{\partial x}(x_n) + \varepsilon u,\ u \sim N(0,1).$$

This is the **Metropolis Adjusted Langevin Algorithm** (**MALA**). If we further add an acceptance step, it will become the Langevin Monte Carlo, a special case of the HMC.

- The **Langevin Monte Carlo** (**LMC**):

$$y = x_n - \frac{\varepsilon^2}{2}\frac{\partial h}{\partial x}(x_n) + \varepsilon u,\ u \sim N(0,1),$$

let $x_{n+1} = y$ with the acceptance probability:

$$\alpha = \min(1, \frac{\pi(y)\phi(u'')}{\pi(x_n)\phi(u)}),\ u'' = u - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(x_n) - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(y),$$

where $\phi(u)$ is the pdf of the $N(0,1)$.

- Or we can rewrite Langevin Monte Carlo to be

$$u' = u - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(x_n),\ u \sim N(0,1),$$
$$y = x_n + \varepsilon u',$$
$$u'' = u' - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(y),$$

let $x_{n+1} = y$ with the acceptance rate α. The above update is called the **leapfrog update**.

- The **Hamiltonian Monte Carlo** is just the Langevin Monte Carlo with multiple leapfrog updates. For each step of HMC:
- Initialize $u \sim N(0,1)$, set the number of leapfrog updates L;
- Do the leapfrog update:

$$u' = u - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(x_n),$$
$$y = x_n + \varepsilon u',$$
$$u'' = u' - \frac{\varepsilon}{2}\frac{\partial h}{\partial x}(y);$$

- If $L > 1$, let $L = L - 1$, $u = u''$, $x_n = y$ and do the above leapfrog update again.
- If $L = 1$, let $x_{n+1} = y$ with the acceptance probability:

$$\alpha = \min(1, \frac{\pi(y)\phi(u'')}{\pi(x_n)\phi(u)}).$$

- What are the advantages of the Hamiltonian Monte Carlo?
- Firstly, if we return to the MALA step

$$x_{n+1} = x_n - \frac{\varepsilon^2}{2}\frac{\partial h}{\partial x}(x_n) + \varepsilon u,\ u \sim N(0,1).$$

  This update allows samples quickly move from the tail to the body, and explore the body after that.
- Secondly, the HMC also allows a high acceptance rate. When $L = 1$, the acceptance rate of the HMC (or LMC) typically should be tuned between 40% and 85%. While the acceptance rate for the Random Walk is around 23.4%.
- Check this **link** for a demonstration of HMC.

Data Augmentation and EM Algorithm

Data Augmentation and EM Algorithm

Problem formulation

- Remember that, generally speaking, there are two ways to solve a problem: 1). **Sampling** from $\pi(x)$, where $\pi(x) \geq 0$ is integrable; 2). Do **optimization** for $f(x)$, where we assume there is a global maximum.
- **Data Augmentation** (**DA**) and the **EM algorithm** further assume that $\pi(x) = \int \pi(x, z)\mathrm{d}z$ and $f(x) = \int f(x, z)\mathrm{d}z$. They talks about how to utilize $\pi(x, z)$ or $f(x, z)$ to solve the original sampling or optimization problems.
- These two algorithms are realized quite differently and they are **incomparable**, but the existence of z is the **common core** of them.
- Some further assumptions include that $\pi(x, z) \geq 0$ is integrable and we have $f(x, z) \geq 0$. We only state the least assumptions here, and some additional conditions need to be satisfied to ensure the convergence.

- Given $\pi(x, z)$, we want to draw samples from $\pi(x)$.
- The original DA algorithm or the **Multiple Imputation** is:
  1. I-step: Draw $\{x_{n,j}\}_{j=1}^{m}$ from $\hat{\pi}_n(x)$; For each $x_{n,j}$, draw $z_{n,j} \sim \pi(z|x_{n,j})$;
  2. P-step: Update that

$$\hat{\pi}_{n+1}(x) = \frac{1}{m} \sum_{j=1}^{m} \pi(x|z_{n,j}).$$

- The standard **DA algorithm** simply use the **Gibbs sampler**:
  1. Draw z_n from $\pi(z|x_n)$;
  2. Draw x_{n+1} from $\pi(x|z_n)$.
- Discarding the samples of z, what remains can be viewed as samples from the $\pi(x)$.
- The **slice sampler** in the tutorial 8 is a typical example of the **DA-based Gibbs sampler**. The HMC can actually be interpreted as a **DA-based Metropolis–Hastings algorithm**.

- Given $f(x, z)$, we want to optimize $f(x)$.
- The standard **EM algorithm** is:
  1. E-step: Calculate that $Q(x|x_n) = \int \log(f(x, z))f(x_n, z)\mathrm{d}z$;
  2. M-step: Obtain $x_{n+1} = \mathrm{argmax}_x\, Q(x|x_n)$.
- The **Generalized EM** (**GEM**) algorithm is:
  1. E-step: Calculate that $Q(x|x_n) = \int \log(f(x, z))f(x_n, z)\mathrm{d}z$;
  2. M-step: Obtain x_{n+1} so that $Q(x_{n+1}|x_n) \geq Q(x_n|x_n)$.

  Use the Newton–Raphson or the gradient ascent in the M-step.
- The **Expectation-Conditional Maximization** (**ECM**) algorithm assumes $x = (x^{(1)}, \ldots, x^{(m)})$ and have:
  1. E-step: Calculate that $Q(x|x_n) = \int \log(f(x, z))f(x_n, z)\mathrm{d}z$;
  2. CM-steps: Optimize $x^{(j)}$ in $Q(x|x_n)$ sequentially to obtain x_{n+1}.

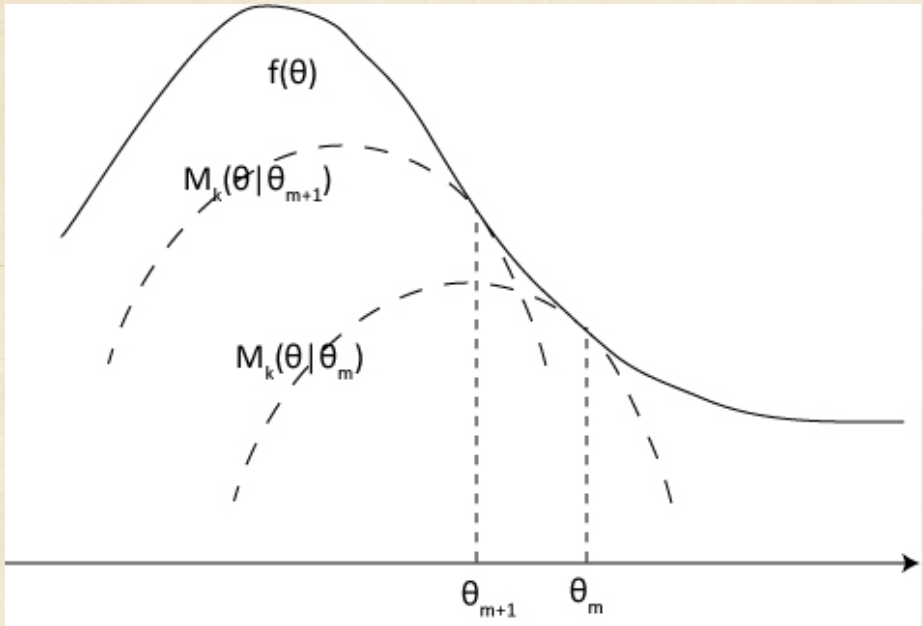
  ECM is also a special case of GEM.
- The **Monte Carlo EM** (**MCEM**) algorithm:
  1. E-step: Calculate that $\hat{Q}(x|x_n) = \sum_{j=1}^{m} \log(f(x, z_j))/m$, $z_j \sim f(z|x_n)$;
  2. M-step: Obtain $x_{n+1} = \mathrm{argmax}_x\, \hat{Q}(x|x_n)$.

MM Algorithm

- Assume that we want to optimize $f(x)$.
- The **minorization–maximization** (**MM**) algorithm is:
  1. Minorization-step: Set $M(x|x_n)$ that minorizes $f(x)$:

$$M(x_n|x_n) = f(x_n),$$
$$M(x|x_n) \leq f(x);$$

  2. Maximization-step: Obtain $x_{n+1} = \operatorname{argmax}_x M(x|x_n)$.



- The **EM algorithm** assumes that $f(x) = \int f(x,z)\mathrm{d}z$, and $f(x,z)$ is nonnegative. To prove that EM is a special case of MM, note that:
- To optimize $f(x)$ is equivalent to optimize $\log(f(x))$.
- To optimize $Q(x|x_n) = \int \log(f(x,z))f(x_n,z)\mathrm{d}z$ is equivalent to optimize

$$M(x|x_n) = \log(f(x_n)) + \int \log(\frac{f(x,z)}{f(x_n,z)})\frac{f(x_n,z)}{f(x_n)}\mathrm{d}z,$$

  which minorizes $\log(f(x))$.
- Proof:

$$M(x_n|x_n) = \log(f(x_n)),$$

$$M(x|x_n) \leq \log(f(x_n)) + \log(\int \frac{f(x,z)}{f(x_n,z)}\frac{f(x_n,z)}{f(x_n)}\mathrm{d}z)$$

$$= \log(f(x_n)) + \log(\frac{f(x)}{f(x_n)}) = \log(f(x)).$$

- The **Quadratic Lower-Bound** (**QLB**) algorithm assumes that the optimization target $f(x)$ is smooth enough.
- If we have a positive definite matrix B such that $\nabla^2 f(x) + B$ is always positive definite, then we have

$$M(x|x_n) = f(x_n) + (x - x_n)^T \nabla f(x_n) - \frac{1}{2}(x - x_n)^T B(x - x_n),$$

  which minorizes $f(x)$.
- Proof:

$$M(x_n|x_n) = f(x_n);$$

  And define $h(x) = f(x) - M(x|x_n)$, observe that

$$\nabla h(x_n) = \nabla f(x_n) - \nabla f(x_n) = \mathbf{0},$$
$$\nabla^2 h(x) = \nabla^2 f(x) + B.$$

  So x_n is the global minimum point and $h(x) \geq h(x_n) = 0$.

- The **Difference of Convex** (**DC**) Programming assumes the optimization target $f(x) = g(x) - g'(x)$, where both $g(x)$ and $g'(x)$ are convex functions.
- We can minorize $f(x)$ by

$$M(x|x_n) = g(x_n) + (x - x_n)^T \nabla g(x_n) - g'(x),$$

which is a concave function and easy to optimize.

- Proof:

$$M(x_n|x_n) = f(x_n);$$

And define
$h(x) = f(x) - M(x|x_n) = g(x) - g(x_n) - (x - x_n)^T \nabla g(x_n)$,
observe that

$$\nabla h(x_n) = \nabla g(x_n) - \nabla g(x_n) = \mathbf{0}.$$

And $h(x)$ is convex. So, x_n is the global minimum point and $h(x) \geq h(x_n) = 0$.

Surrogate Methods

Surrogate Methods
Problem formulation

- Assuming that we want to optimize $f(x)$, the **surrogate method** tries to replace the optimization target $f(x)$ by a surrogate optimization target $S(x|x_{1:n})$ that may depends on all or part of the historical iterations and can help us easier find x_{n+1}.
- Obviously the **MM algorithm** is a special case of the surrogate method, which means $S(x|x_{1:n}) = M(x|x_n)$.
- Actually, most optimization algorithms can be interpreted as the surrogate method. For example, the **Newton–Raphson method** simply assumes the surrogate function to be the **local second order approximation** of $f(x)$:

$$S(x|x_{1:n}) = f(x_n) + (x - x_n)^T \nabla f(x_n) + \frac{1}{2}(x - x_n)^T \nabla^2 f(x_n)(x - x_n).$$

  And the **gradient method** assume the surrogate function to be the **local first order approximation**:

$$S(x|x_{1:n}) = f(x_n) + (x - x_n)^T \nabla f(x_n).$$

- The '**surrogate**' in optimization is like the '**proposal**' in sampling.

- Assuming that we want to optimize $f(x)$, the **Bayesian Optimization** (**BO**) use the Gaussian process to model the target function $f(x)$ and construct the surrogate function $S(x|x_{1:n})$, which is called the **acquisition function** in BO.
- A **Gaussian process** (**GP**) is a distribution over a space of functions. It can be used to model functions which are unknown or obscured by stochastic noise. (Remember that the **Dirichlet process** is a distribution of distributions.)
- A Gaussian process is completely specified by its **mean function** $\mu_0(x)$ and **covariance function** or **kernel** $\Sigma_0(x, x')$.
- If $f(x) \sim \mathcal{GP}(\mu_0, \Sigma_0)$, for $x_{1:n} = [x_1, \ldots, x_n]$,

$$f(x_{1:n}) \sim N(\mu_0(x_{1:n}), \Sigma_0(x_{1:n}, x_{1:n}))$$

where $f(x_{1:n}) = [f(x_1), \ldots, f(x_n)]$, $\mu_0(x_{1:n}) = [\mu_0(x_1), \ldots, \mu_0(x_n)]$, $\Sigma_0(x_{1:n}, x_{1:n}) = [\Sigma_0(x_i, x_j)]_{n \times n}$.

- Here is a simple example of the GP:

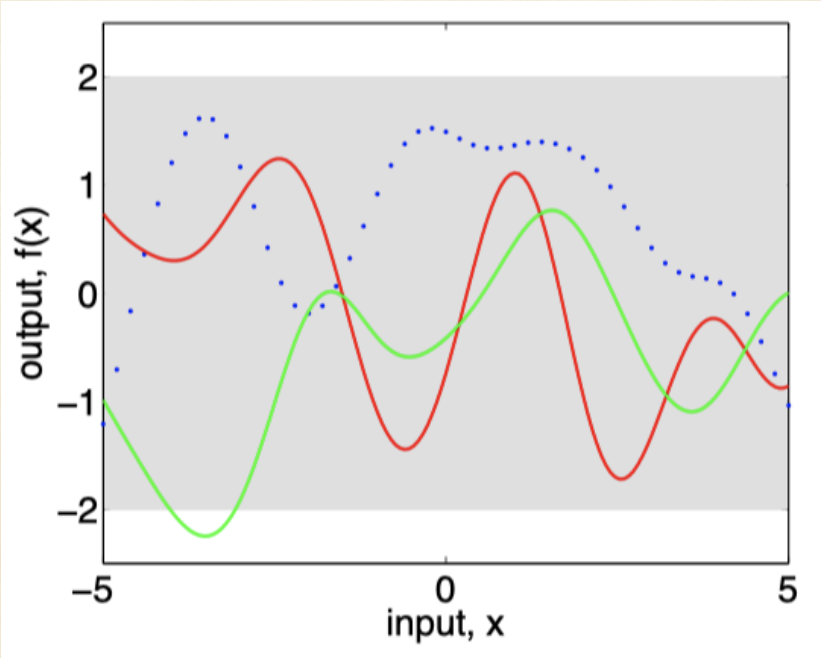


Figure: Three samples and the 95% confidence region, when $\mu_0(x) = 0$, $\Sigma_0(x, x') = 1$.

- $\mathbb{E}(f(x)) = \mu_0(x) = 0$.
- $\mathrm{Var}(f(x)) = \sigma_0^2(x) = \Sigma_0(x, x) = 1$.

- The Gaussian process can be use as a **conjugate prior**.
- Set the **GP prior** $\mathcal{GP}(\mu_0, \Sigma_0)$ for $f(x)$.
- For any $x, x', x_{1:n}$, there is

$$[f(x), f(x'), f(x_{1:n})] \sim N(\mu_0([x, x', x_{1:n}]), \Sigma_0([x, x', x_{1:n}], [x, x', x_{1:n}])).$$

- So $f(x)|f(x_{1:n})$ and $f(x), f(x')|f(x_{1:n})$ would also follow normal distributions, which result in the **GP posterior** $\mathcal{GP}(\mu_n, \Sigma_n)$ where

$$\begin{aligned}
\mu_n(x) &= \mathbb{E}(f(x)|f(x_{1:n})) \\
&= \mu_0(x) + \Sigma_0(x, x_{1:n})\Sigma_0(x_{1:n}, x_{1:n})^{-1}(f(x_{1:n}) - \mu_0(x_{1:n})), \\
\Sigma_n(x, x') &= \mathrm{Cov}(f(x), f(x')|f(x_{1:n})) \\
&= \Sigma_0(x, x') - \Sigma_0([x, x'], x_{1:n})\Sigma_0(x_{1:n}, x_{1:n})^{-1}\Sigma_0(x_{1:n}, [x, x']).
\end{aligned}$$

- The above updates are also called the **GP regression**.

- Here is a simple example of the GP regression:

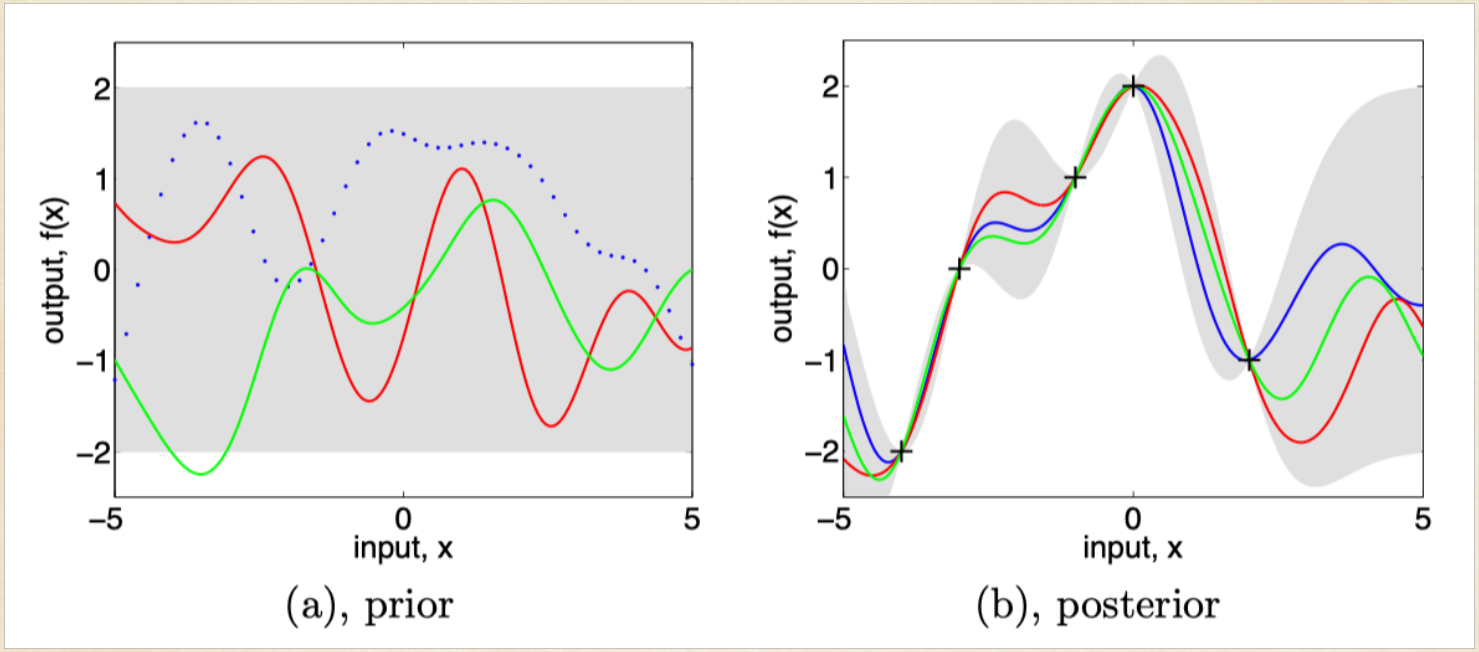


Figure: Three samples for prior or posterior and the corresponding 95% confidence regions.

- Check this **link** for an online demonstration of the GP regression.

- The **Bayesian Optimization** deals with the problem:

$$\max_{x \in A} f(x)$$

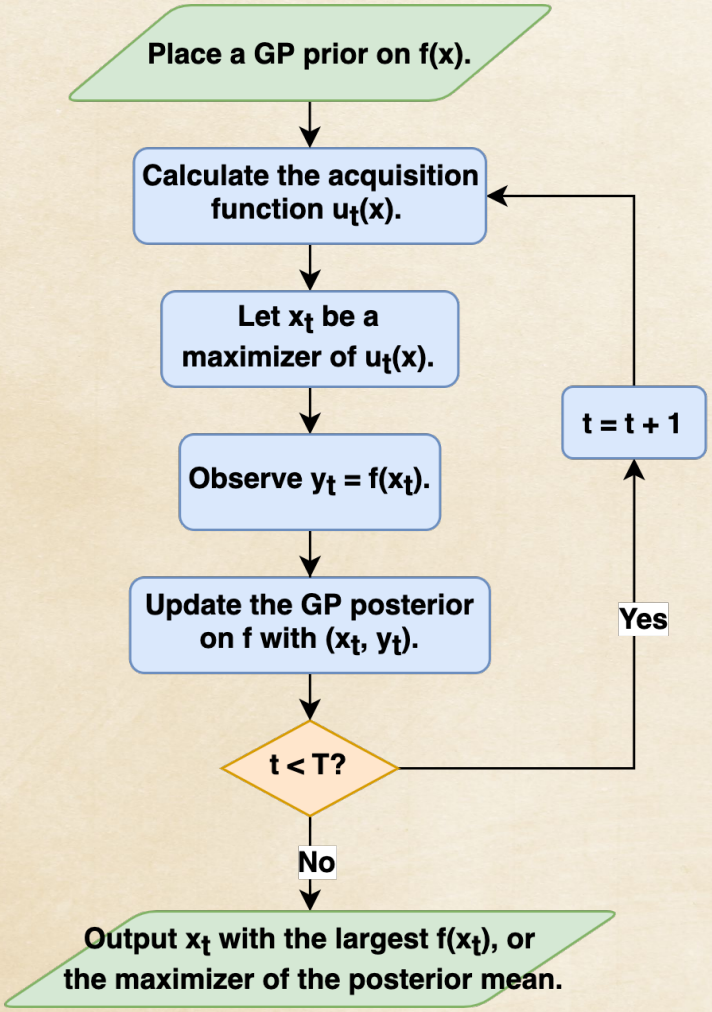
  where
  - x: $d \leq 20$;
  - A: simple set (rectangle, simplex);
  - $f(x)$: **black box** (extremely difficult to evaluate, no concavity or linearity, sometime maybe obscured by stochastic noise, in most cases continuous).
- A typical example is to tune the **hyperparameters** of a complicated algorithm. In this case, A is a hyperrectangle if we set a range for each hyperparameter, and $f(x)$ is a function that evaluate the performance of the algorithm, like a gain function or a negative loss function.

- Different from the normal optimization problem, BO has two goal:
  - **Exploration**: learn how $f(x)$ looks like;
  - **Exploitation**: optimize $f(x)$.
- BO generally wants to use the first bullet to assist in achieving the second. Remember that the sampling methodology is good at exploration, but as $f(x)$ is expensive to evaluate, BO has to resort to the Bayesian methodology, the Gaussian process.
- To be more specifically, the **ultimate goal** of BO is to:
  - Reduce the **regret**:

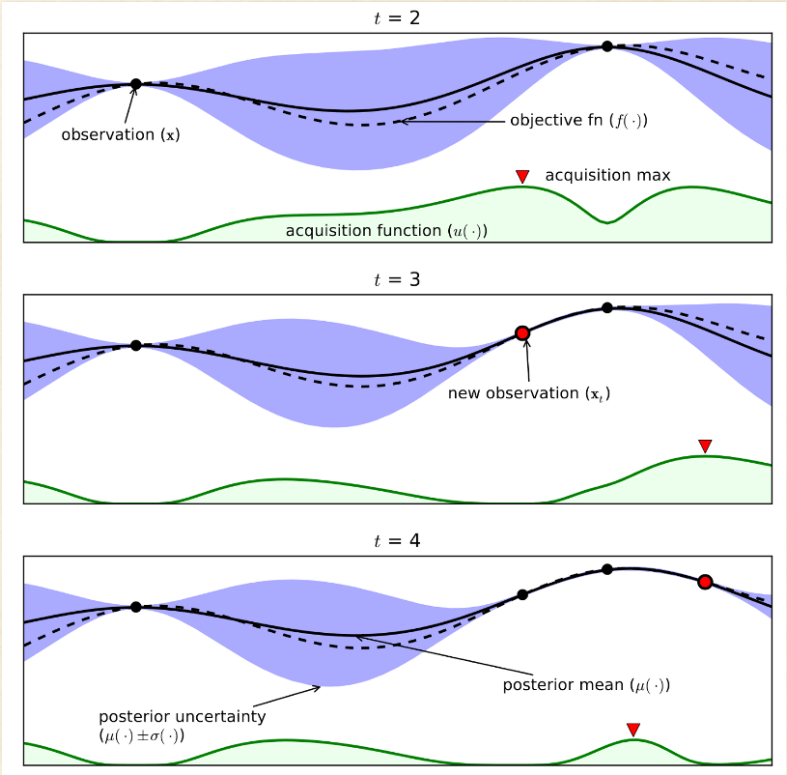
$$\mathcal{R}(T) = \sum_{t=1}^{T} (f(x^*) - f(x_t)),$$

    where x^* is the maximizer of $f(x)$, T is the total iteration time.
  - Or construct algorithms such that $\mathcal{R}(T)$ is **sublinear**.

Surrogate Methods
Bayesian Optimization



(a) Bayesian Optimization algorithm



(b) example (Upper Confidence Bound)

Surrogate Methods
Bayesian Optimization

- In practice, the **QMC** in tutorial 10 may be used to do the initialization for the Bayesian Optimization.
- In the nth iteration, we have the posterior distribution $\mathcal{GP}(\mu_n, \Sigma_n)$, from which there are two fundamental **acquisition functions**:
  - **Upper Confidence Bound**: $u_n(x) = \mu_n(x) + \beta_n \sigma_n(x)$, where $\sigma_n(x) = \sqrt{\Sigma_n(x, x)}$.
  - **Thompson Sampling**: draw $u_n(x) \sim \mathcal{GP}(\mu_n, \beta_n^2 \Sigma_n)$.
- β_n serves as a compromise between the exploration and the exploitation and sometimes we let it depends on the iteration number.
- Here, an acquisition function is just a **surrogate function** $S(x|x_{1:n}) = u_n(x)$. Note that, this surrogate function depends on all the previous iterations because the GP posterior depends on all the previous observations.
- Remember that, just like the Annealing, the Thompson Sampling is another **sampling-based optimization** algorithm.

References

- Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. arXiv preprint arXiv:1701.02434.
- Horst, R., & Thoai, N. V. (1999). DC programming: overview. Journal of Optimization Theory and Applications, 103(1), 1-43.
- Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

Thanks!