$\LaTeX$ command declarations here.

AERSP 597 - Machine Learning in Aerosapce Engineering

Lecture 12, Gaussian Process Regression: Bayesian Optimization

Instructor: Daning Huang

In [1]:
```
from __future__ import division
from warnings import filterwarnings
filterwarnings('ignore')

import random
import numpy as np
import scipy as scp
import scipy.stats
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

TODAY: Gaussian Process Regression - V

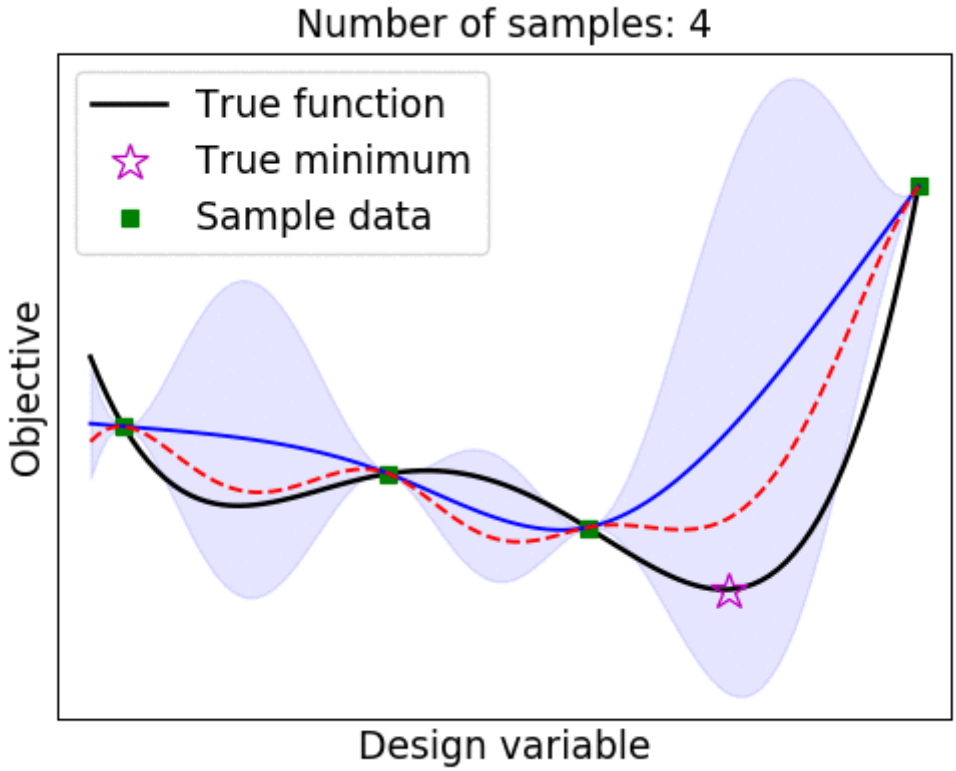
Bayesian Optimization

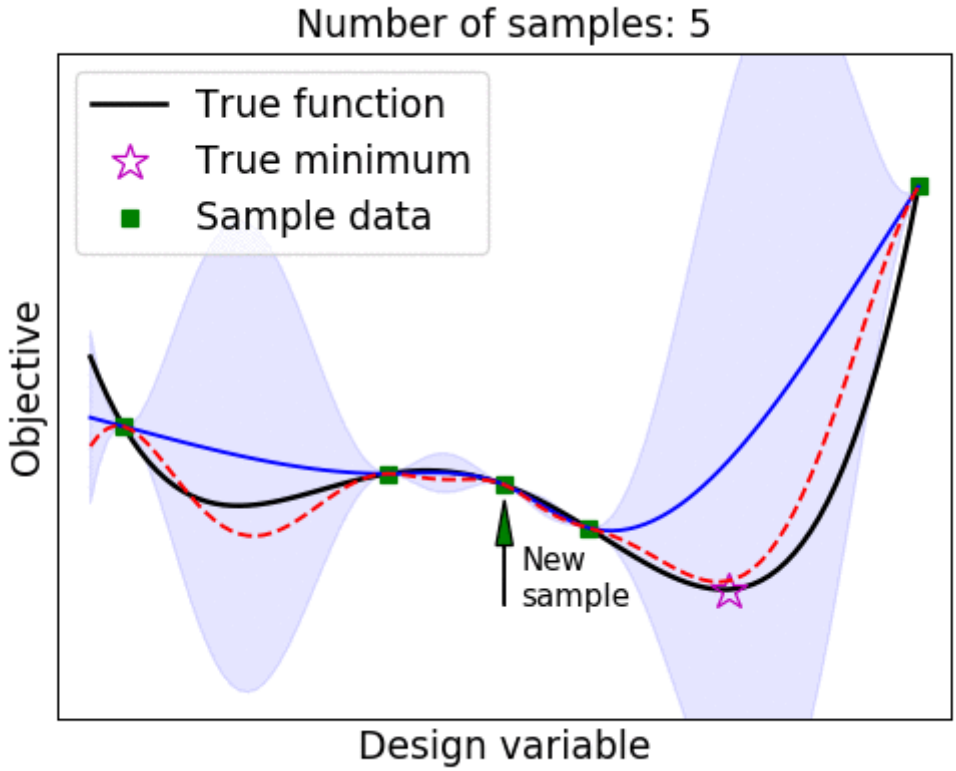
- Surrogates
- Acquisition function
- Inner optimization
- Updating schemes

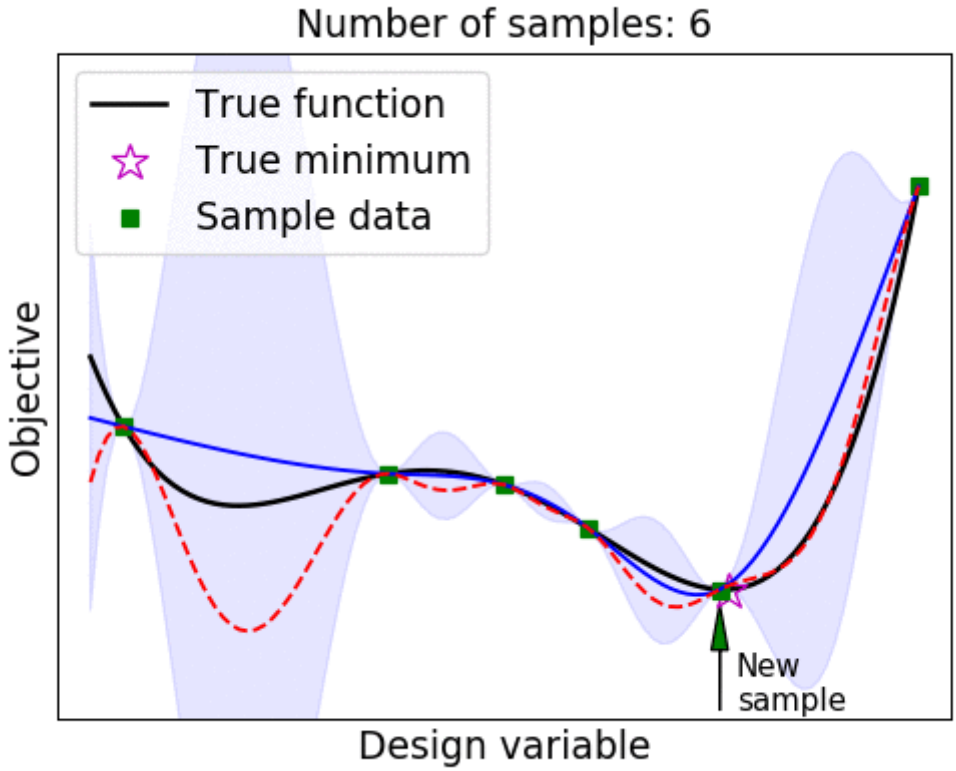
References

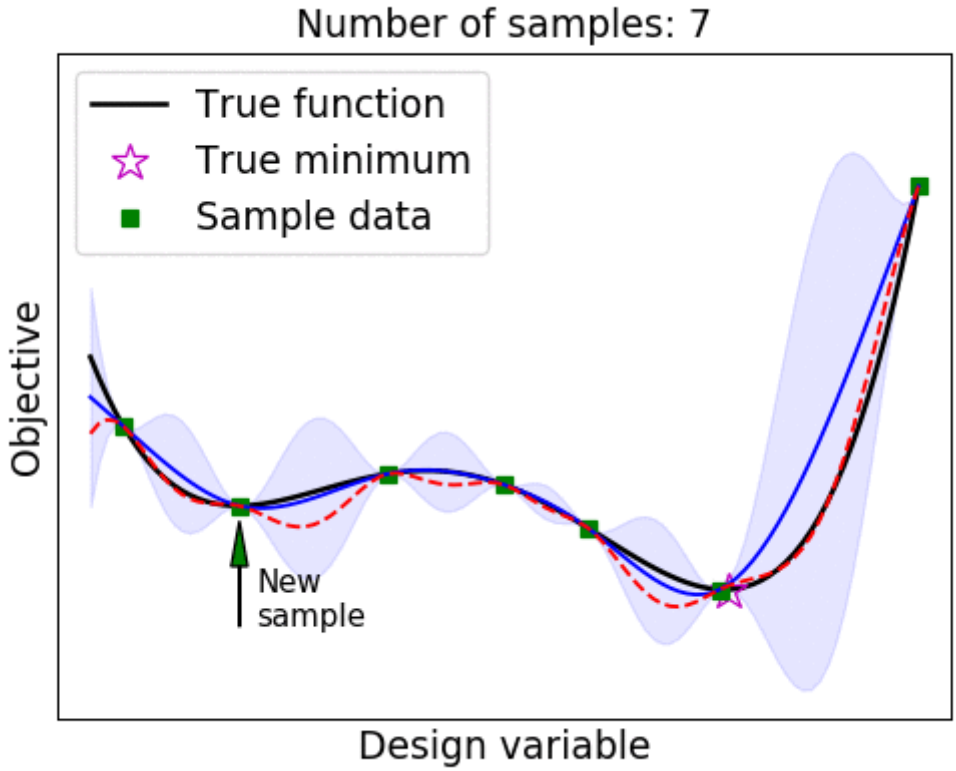
- Engineering Design via Surrogate Modelling: A Practical Guide, by A. Forrester, A. Sobester, A. Keane

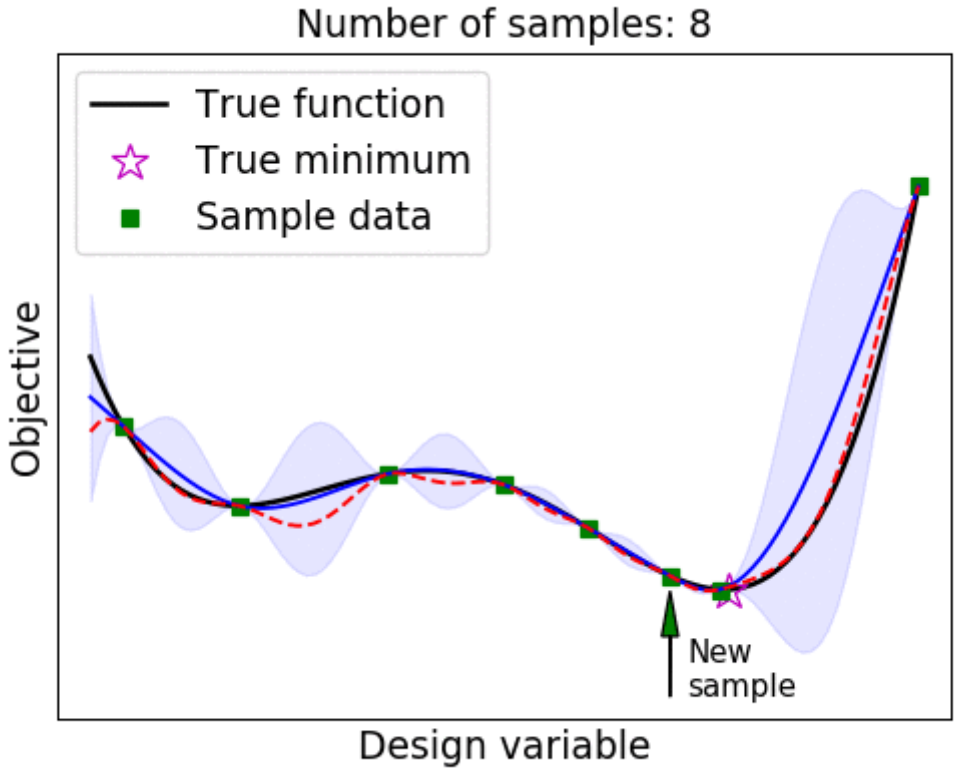
An example

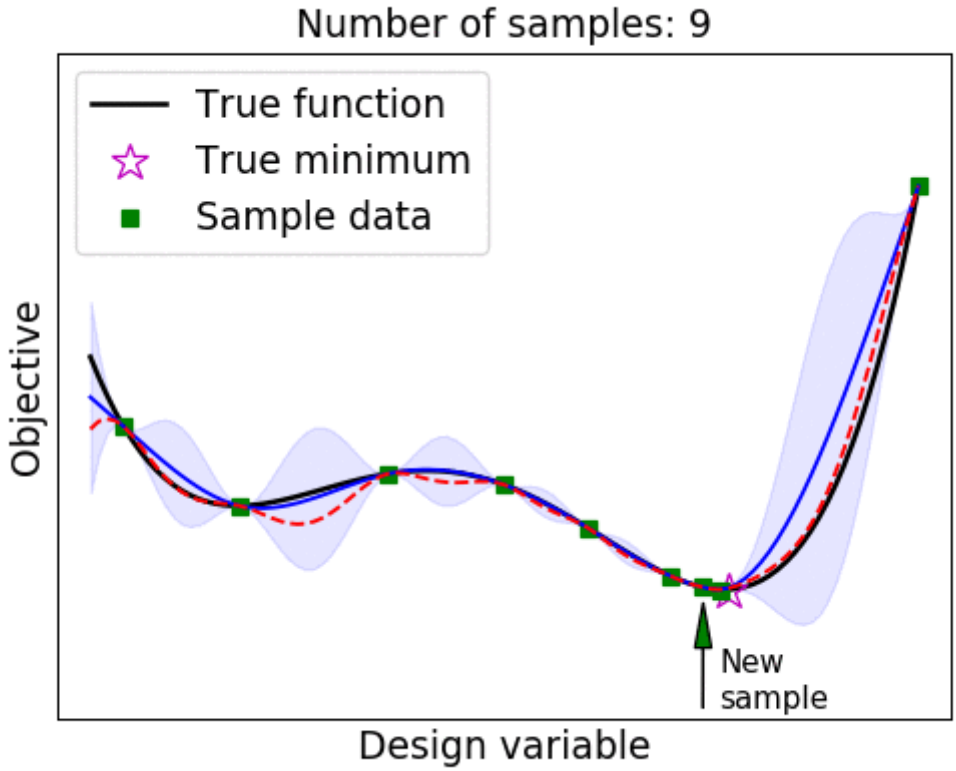

Number of samples: 4
True function
True minimum
Sample data
Objective
Design variable

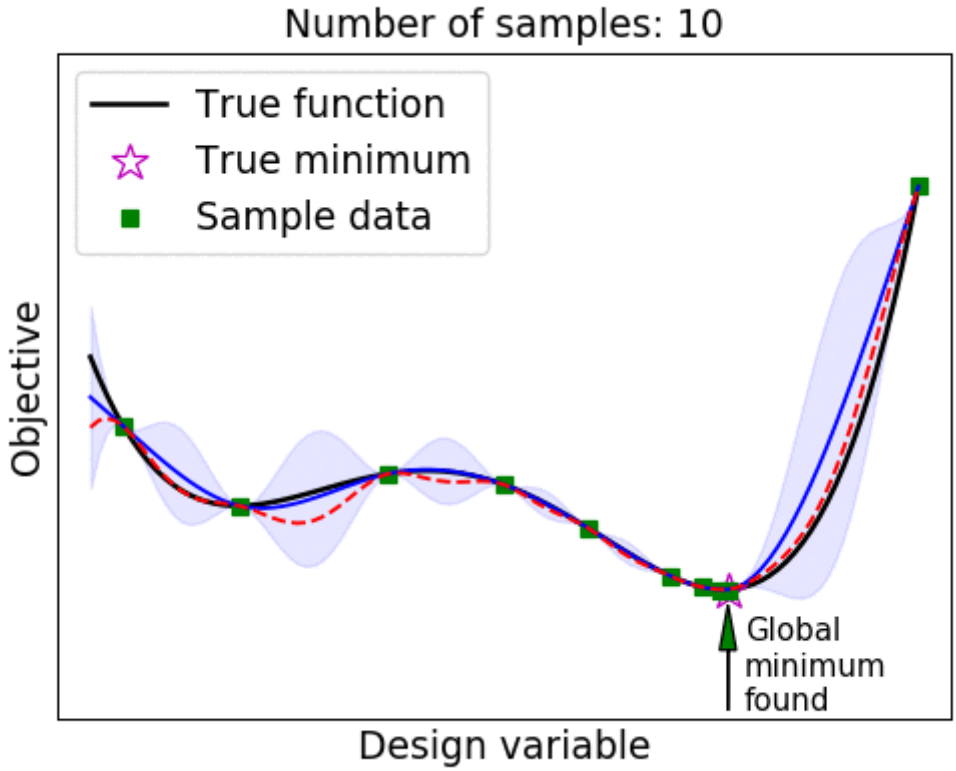

Number of samples: 5
True function
True minimum
Sample data
New
sample
Objective
Design variable


Number of samples: 6
True function
True minimum
Sample data
Objective
New
sample
Design variable


Number of samples: 7
True function
True minimum
Sample data
Objective
New
sample
Design variable


Number of samples: 8
True function
True minimum
Sample data
Objective
New
sample
Design variable


Number of samples: 9
True function
True minimum
Sample data
Objective
New
sample
Design variable



Introduction

The problem

The objective function is expensive to compute when it is, for example, the lift-to-drag ratio of an aircraft wing using a computational fluid dynamics (CFD) solver, or nonlinear deformation of composite structure using a finite element (FE) solver. The objective function is impossible to compute when it is, for example, data obtained from experiments.

The optimization problem is written formally as,

$$\begin{aligned} \arg\min_{\mathbf{d}} \quad & J(\mathbf{d};\mathbf{U}) \\ \text{s. t.} \quad & \mathbf{c}_E(\mathbf{d};\mathbf{U}) = 0 \\ & \mathbf{c}_I(\mathbf{d};\mathbf{U}) \geq 0 \end{aligned}$$

where $\mathbf{d}$ is the vector of design variables, $\mathbf{c}_E$ and $\mathbf{c}_I$ are the aggregated equality and inequality constraint functions, respectively. The state variables $\mathbf{U}$ are those implicitly involved in the evaluation of the objective and constraint functions. Examples of $\mathbf{U}$ are the flow variables in a CFD solver or the displacement field in the structural FE solver.

In above cases, in order to solve the optimization problem during a practical time period, the number of evaluation of the objective function has to be limited in the optimization algorithm. There are two limitations on the optimization algorithm:

- The design space cannot be explored by carrying out numerous *direct* evaluation of the objective function;

- The derivative of the objective function w.r.t. design variables cannot be computed using the finite difference approach.

These limitations prevent the direct application of any gradient-free or gradient-based algorithms to the optimization problem.

Surrogate-based optimization

To overcome the limitations, one approach is to use surrogate-based optimization (SBO). The SBO algorithms typically contain two key ingredients, **a surrogate model** and **an acquisition function**. The surrogate model is employed to approximate the expensive objective function. Since the surrogate is computationally efficient, it allows for fast evaluation of approximated objective function as well as its derivative w.r.t. the design variables. The acquisition function is a criterion for selecting the points in the design space that is potentially a solution to the optimization problem. The acquisition function is designed to take into account of both **exploration**, i.e. sampling from areas of high uncertainty, and **exploitation**, i.e. sampling from areas likely to contain the minimizer of the objective function.

The general procedure of SBO is as follows,

1. [**Initialize dataset**] Generate a sample data set $\mathcal{D} = \{(\mathbf{d}_i, J_i, \mathbf{c}_{Ei}, \mathbf{c}_{Ii})\}_{i=1}^{N_s}$ by evaluating the objective and constraint functions $J_i, \mathbf{c}_{Ei}, \mathbf{c}_{Ii}$ at a few sample points $\mathbf{d}_i$ in the design space.
2. [**Initialize surrogate**] Generate surrogates $J^{sur}(\mathbf{d})$, $\mathbf{c}_E^{sur}(\mathbf{d})$, $\mathbf{c}_I^{sur}(\mathbf{d})$ using the sample data set $\mathcal{D}$.
3. [**Inner Optimization**] Find the design point $\mathbf{d}^*$ by solving an optimization problem that consists of the surrogates and an acquisition function C.
$$\begin{aligned} \arg\min_{\mathbf{d}} \quad & C(J^{sur}, \mathbf{d}) \\ \text{s. t.} \quad & \mathbf{c}_E^{sur}(\mathbf{d}) = 0 \\ & \mathbf{c}_I^{sur}(\mathbf{d}) \geq 0 \end{aligned}$$
4. [**Acquire new data**] Evaluate the objective and constraint functions $J^*, \mathbf{c}_E^*, \mathbf{c}_I^*$ at the new design point $\mathbf{d}^*$.
5. [**Update surrogate**] Update the surrogate using the sample data set augmented with the new design point $\mathcal{D}^* = \mathcal{D} \cup \{(\mathbf{d}^*, J^*, \mathbf{c}_E^*, \mathbf{c}_I^*)\}$.
6. [**Loop**] Repeat steps 3-5 until the convergence or stopping conditions are reached.

One category of SBO is the one-shot approach, which only contains steps 1-4. In the one-shot approach, the acquisition function is the surrogate itself. The issue is that the initial sample set does not necessarily represent well the distribution of the objective function over the whole design space. As a result, the optimal solution found by the algorithm may be far off from the true value of objective function. The other category of SBO is the updating approach that, of course, contains all the six steps. This is the category where the Bayesian optimization lies.

Bayesian optimization

Concepts

Bayesian optimization (BO) is essentially the six-step SBO procedure with a statistical interpretation. Adapted from wikipedia (https://en.wikipedia.org/wiki/Bayesian_optimization): The expensive function is treated as a random function with a prior distribution, which captures the beliefs about the behavior of the function. The prior distribution is updated using the data set to form the the posterior distribution over the objective function. The posterior distribution is used to construct an acquisition function that determines the next design point.

The two precursors of BO are Kushner1964 (http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1431594) and Mockus1975 (https://link.springer.com/chapter/10.1007%2F978-3-662-38527-2_55). A good review of BO is provided in Brochu2010 (https://arxiv.org/abs/1012.2599). The Gaussian process regression (GPR) model is a popular choice for the surrogate model in BO. Some outstanding the GPR-based BO algorithms include, but not limited to,

1. BayesOpt: code (https://github.com/rmcantin/bayesopt), doc (https://rmcantin.bitbucket.io/html/), paper (https://arxiv.org/abs/1405.7430).
2. HIPS/spearmint: code (https://github.com/HIPS/Spearmint).
3. GPyOpt: code (https://github.com/SheffieldML/GPyOpt), doc (http://sheffieldml.github.io/GPyOpt/)
4. GPflowOpt: code (https://github.com/GPflow/GPflowOpt), doc (https://gpflowopt.readthedocs.io/en/latest/), paper (https://arxiv.org/abs/1711.03845)

Besides GPR, there are BO algorithms using other surrogate models, such as those in SMAC (code (http://www.cs.ubc.ca/labs/beta/Projects/SMAC/), paper (https://www.cs.ubc.ca/~hutter/papers/10-TR-SMAC.pdf)). A list of implementations of BO is provided here (https://rmcantin.bitbucket.io/html/relsoft.html). Finally, note that in the engineering community, the BO algorithm tends to appear by the name efficient global optimization (EGO) algorithm Jones1998 (https://link.springer.com/article/10.1023/A:1008306431147), Sasena2002 (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.4697&rep=rep1&type=pdf).

Next, the key components of BO, the surrogate model and the acquisition function, as well as the treatment of constraints will be discussed.

Surrogate model

In this and the following sections, the discussion will be based on the GPR model. The prediction of the GPR model at a new design point $\mathbf{d}^*$ follows a Gaussian probability distribution,

$$J^* = J^{sur}(\mathbf{d}^*) \sim \mathcal{N}(\mu(\mathbf{d}^*), \sigma^2(\mathbf{d}^*)), \quad \text{or} \quad P(J^{sur} = J^*) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(J^* - \mu)^2}{2\sigma^2}\right)$$

where μ is the predicted value of objective function and σ^2 is the variance, i.e. the uncertainty of the prediction. The details of GPR models have been discussed in the previous articles and will be skipped.

Acquisition function for unconstrained optimization

The BO is initially developed for unconstrained problems. Surveys of acquisition functions for such problems can be found in, for example, [Sasena2002] and Gelbart2015 (https://dash.harvard.edu/handle/1/17467236). There are three basic acquisition functions: (1)

Probability Improvement (PI) [Kushner1964], (2) Expected Improvement (EI) Mockus1994 (https://link.springer.com/article/10.1007/BF01099263), Lizotte2008 (http://www.csd.uwo.ca/~dlizotte/publications/lizotte_phd_thesis.pdf), (3) Lower Confidence Bound (LCB) Cox1992 (https://ieeexplore.ieee.org/document/271617/). Besides the basic ones, there are acquisition functions based on Thompson Sampling and the information theory (entropy). In general, the acquisition functions are evolving towards (1) finding multiple candidate points in one iteration, (2) taking advantage of parallel computing via techniques like asynchronization. Nonetheless, this article will focus only on the three basic acquisition functions.

PI

Defined as the probability of the new design point $\mathbf{d}^*$ to offer a better value of the objective function J^* than the minimum objective function in the sample data set $J_{\min} = \min\{J_i\}$.

$$C_{PI}(\mathbf{d}^*) = P(J^* \leq J_{\min})$$

EI

Defined as the expectation of the improvement in the objective function at the new design point. In the literature, EI has been generalized to include user-specified parameters that control the exploitation-exploration trade-off. The generalized EI is written as,

$$C_{EI}(\mathbf{d}^*) = \mathbb{E}[I(\mathbf{d}^*)], \quad I(\mathbf{d}) = \max(0, (J_{\min} - J^{sur}(\mathbf{d}) - \zeta\sigma(\mathbf{d}))^g)$$

where $\zeta \geq 0$ and $g \geq 1$ are the user-specified parameters. The classical form of EI is obtained with $\zeta = 0$ and $g = 1$. A larger g or ζ will put more weight on the exploration. Note that when $\zeta = 0$ and $g = 0$, EI reduces to PI. For a GPR model, the closed-form expression is available for C_{EI}. For the case $g = 1$,

$$\begin{aligned} C_{EI}(\mathbf{d}^*) &= \int_0^\infty I P(J^{sur} = J_{\min} - \zeta\sigma - I) dI \\ &= \begin{cases} \sigma[z\Phi(z) + \phi(z)], & \sigma > 0 \\ 0, & \sigma = 0 \end{cases} \end{aligned}$$

where $z = \frac{J_{\min} - \zeta\sigma - \mu}{\sigma}$, and Φ and ϕ are the cummulative distribution and probability density functions, respectively.

LCB

Defined using the LCB concept,

$$C_{LCB}(\mathbf{d}^*) = \mu(\mathbf{d}^*) - \zeta\sigma(\mathbf{d}^*)$$

where the probability of $J^{sur} < C_{LCB}$ is a constant controlled by the user-specified parameter $\zeta > 0$. A larger ζ will put more weight on the exploration. The gradient of C_{LCB} w.r.t. $\mathbf{d}$ is straight-forward.

Discussion

The PI acquisition function is purely exploitation, which is undesirable for global optimization. The EI and LCB acquisition functions are high when J^* approaches the optimum point, or the uncertainty of J^* is high. Therefore, both C_{EI} and C_{LCB} achieve a balance between exploitation and exploration.

The exploitation-exploration trade-off of EI and LCB functions can be further tuned by the cooling scheme. In the cooling scheme, the optimization starts with a large user-specified parameter for more exploration and gradually decreases the parameter to focus on exploitation. However, the effect of this scheme is controversial [Sasena2002] and [Brochu2010].

Treatment of constraints

The acquisition functions discussed in the previous section work well with optimization problems without constraints or with box constraints only. For more complex constraints, there are two types of treatments. The first type is the direct approach. As suggested in [Sasena2002], the objective and constraint functions are replaced with the surrogates, and the following problem is solved,

$$\begin{aligned} \arg\min_{\mathbf{d}} \quad & C(\boldsymbol{J}^{sur}, \mathbf{d}) \\ \text{s. t.} \quad & \mathbf{c}_E^{sur}(\mathbf{d}) = 0 \\ & \mathbf{c}_I^{sur}(\mathbf{d}) \geq 0 \end{aligned}$$

Similar to [Sasena2002] is the expected violation (EV) method Audet2000 (https://arc.aiaa.org/doi/pdf/10.2514/6.2000-4891): The optimization is carried out by filling the design space using latin hypercube sampling and filtering out infeasible candidate points using the so-called EV function, which takes the same form of EI. Essentially, the constraint surrogates $\mathbf{c}^{sur}$ are replaced by $C_{EV}(\mathbf{c}^{sur}, \mathbf{d})$.

The other type is the indirect approach, which is further divided into two categories, as discussed in [Gelbart2015]. In the first category, the constrained problem is converted to an unconstrained one using classical (non-BO) methods. Three representative approaches are,

1. Barrier methods: The iterates are *prevented* from leaving the feasible region by augmenting the objective function with a barrier function, which causes the objective to grow to infinity at the boundary of the feasible region.
2. Penalty methods: The objective function is augmented with a penalty term, which *still allows* the iterates to leave the feasible region but the iterates become feasible as the penalty increases to infinity in the process of optimization.
3. Augmented Lagrangian methods: The problem is solved via the Lagrangian form augmented with the extra term in penalty methods, while the Lagrangian multipliers are controlled by the penalty parameters.

In the second category of indirect approach, the acquisition function is modified to incorporate the constraint surrogates. two representative approaches are,

1. Modified EI methods: An example is the EI with constraints (EIC) Schonlau1998 (https://www.jstor.org/stable/4356058). The EI for the objective function is augmented to take account of the constraints. In EIC, the EI is multiplied by the probability that the constraints are satisfied, so that the improvement (almost) only occurs at feasible candidate points.
2. Marginalization integral (MI) methods: Examples are integrated expected conditional improvement Gramacy2011 (https://www.researchgate.net/publication/45913547_Optimization_Under_Unknown_Constraint and expected volume minimization Picheny2014 (http://proceedings.mlr.press/v33/picheny14.pdf). The acqusition function at $\mathbf{d}$ is defined using an integral over the whole design space,

$$C_{MI}(\mathbf{d}) = \int F(\mathbf{d}, \mathbf{d}')h(\mathbf{d}')d\mathbf{d}'$$

where $F(\mathbf{d}, \mathbf{d}')$ is a measure of improvement at $\mathbf{d}'$ given observation at $\mathbf{d}$ and $h(\mathbf{d}')$ is the probability that the constraints are satisfied at $\mathbf{d}'$. The maximizer of C_{MI} provides the best *overall* improvement in the design space. The inclusion of $h(\mathbf{d})$ encourages picking candidate points likely to be feasible, while *still permits* the picking of points in infeasible region that may provide improvement over the whole design space.

In terms of programming, the direct approach is probably the easiest one to implement. Particularly, the non-BO indirect approaches may be applied to solve the subproblems derived from the the direct approach, if standard packages for numerical optimization are employed. The challenge in the MI methods is the integration over the design space, which could be intractable in higher dimensions.

Inner optimization step

With the surrogates and the acquisition function ready, one then proceeds to the inner optimization step.

Choice of algorithm

No matter which algorithm is used, eventually the BO process boils down to a series of non-convex optimization subproblems. It is hard to find the global optimum of a non-convex function. One choice is to use the evolutionary algorithms, or some so-called global algorithms like the DIRECT, which are claimed to be able to find the global optimum given sufficiently many iterations. Another choice is to apply the algorithms for convex optimization, especially the gradient-based ones, with multiple restarts.

Sometimes the latter is preferred. Both choices *do not* guarantee the convergence to global optimum. The convergence rate of the former is much slower than the latter, especially in high-dimensional space. In practice, multiple restarts usually result in a good global sub-optimal point that is sufficient for the engineering purposes. A merit of the former, though, is that some algorithms can handle disjoint feasible regions.

Gradients

If a gradient-based algorithm is employed in the inner optimization step, it is necessary to compute the gradients of the surrogate and the acquisition function analytically (or using automatic differentiation), instead of using finite difference. The latter could destabilize the iterations of gradient descent near some "singularity" points at which the GP model is ill-defined.

For example, the gradient of C_{EI} w.r.t. $\mathbf{d}$ for $\sigma > 0$ is,

$$\frac{\partial C_{EI}}{\partial \mathbf{d}} = -\frac{\partial \mu}{\partial \mathbf{d}}\Phi(z) + \frac{\partial \sigma}{\partial \mathbf{d}}[\phi(z) - \zeta\Phi(z)]$$

where one needs to know $\frac{\partial \mu}{\partial \mathbf{d}}$ and $\frac{\partial \sigma}{\partial \mathbf{d}}$.

Note that, in the trivial treatment of the multiple output case, the process variances σ_f^2 of the output are determined independently. Therefore, it is sufficient to consider the single output case. The mean and variance at a single point are, respectively,

$$\begin{aligned} m(\mathbf{x}) &= \bar{\mathbf{g}}^T \mathbf{k}_u(\mathbf{x}) + \bar{\mathbf{b}}^T \mathbf{h}_u(\mathbf{x}) \\ \sigma^2(\mathbf{x}) &= \sigma_f^2 - [\mathbf{L}^{-1}\mathbf{k}_u(\mathbf{x})]^2 + [\mathbf{R}^{-T}[\mathbf{h}_u(\mathbf{x}) - \mathbf{F}^T[\mathbf{L}^{-1}\mathbf{k}_u(\mathbf{x})]]]^2 \\ &\equiv \sigma_f^2 - \mathbf{e}_1^T\mathbf{e}_1 + \mathbf{e}_2^T\mathbf{e}_2 \end{aligned}$$

where $\mathbf{k}_u = \mathbf{K}_{su}$ and $\mathbf{h}_u = \mathbf{H}_u^T$ and

$$\begin{aligned} \bar{\mathbf{b}} &= \mathbf{R}^{-1}[\mathbf{Q}^T(\mathbf{L}^{-1}\mathbf{y}_s)] \\ \bar{\mathbf{g}} &= \mathbf{L}^{-T}[\mathbf{L}^{-1}(\mathbf{y}_s - \mathbf{H}\bar{\mathbf{b}})] \end{aligned}$$

The gradient of the mean is computed in a straight-forward manner,

$$\frac{\partial m}{\partial \mathbf{x}} = \bar{\mathbf{g}}^T \frac{\partial \mathbf{k}_u}{\partial \mathbf{x}} + \bar{\mathbf{b}}^T \frac{\partial \mathbf{h}_u}{\partial \mathbf{x}}$$

where $\frac{\partial \mathbf{k}_u}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{h}_u}{\partial \mathbf{x}}$ are obtained from the regression and mean functions, respectively.

The computation of the gradient of the variance is a little bit more involved,

$$\begin{aligned} \frac{\partial \sigma^2}{\partial \mathbf{x}} &= -2\mathbf{e}_1^T\mathbf{L}^{-1}\frac{\partial \mathbf{k}_u}{\partial \mathbf{x}} + 2\mathbf{e}_2^T\mathbf{R}^{-T}\left(\frac{\partial \mathbf{h}_u}{\partial \mathbf{x}} - \mathbf{F}^T\mathbf{L}^{-1}\frac{\partial \mathbf{k}_u}{\partial \mathbf{x}}\right) \\ &= -2[\mathbf{L}^{-T}(\mathbf{F}\mathbf{r} + \mathbf{e}_1)]^T\frac{\partial \mathbf{k}_u}{\partial \mathbf{x}} + 2\mathbf{r}^T\frac{\partial \mathbf{h}_u}{\partial \mathbf{x}} \end{aligned}$$

where $\mathbf{r} = \mathbf{R}^{-1}\mathbf{e}_2$. The computation is organized so as to minimize the number of linear solves of the triangular matrices.

Update surrogate step

Relearn rate of the surrogate

The training of the surrogates could become expensive as the number of samples increases. Also, contrary to intuition, updating the hyperparameters of the surrogate at every iteration may be unnecessary or even counterproductive Bull2011 (http://www.jmlr.org/papers/v12/bull11a.html). A better strategy is to introduce a *relearn rate*: Update the sample data set in the surrogate every iteration, but update the hyperparameters every *few* iterations.

In practice, the effectiveness of this strategy depends on the quality of the surrogate implementation. If the current hyperparameters are far off from the converged values, keeping the current hyperparameters would reduce the convergence rate of the whole BO algorithm.

Updating scheme for GPR

Formal, we set up the problem like this: Consider a scenario where a GP model is trained using a large sample data set (N points), and now suppose a few new samples (M points, $M \ll N$) are to be appended to the data set. The question is, if the existing hyperparameters are kept, how to recomputing the coefficient matrices in GPR, including $\mathbf{L}$, $\mathbf{F}$, $\mathbf{Q}$ and $\mathbf{R}$, efficiently.

Strictly speaking, the addition of new samples will result in the recomputation of everything, even though the hyperparameters remain the same: The std and mean of the samples are modified, so is the correlation matrix $\mathbf{K}$, and so are the follow-up matrix decompositions. However, now that $M \ll N$, one can assume that the std and mean of the new sample data set are the same as those of the old one. As a result, the portion of $\mathbf{K}$ associated with the old sample data set remains the same, and the new coefficient matrices can be computed via *partial* matrix decompositions, which could save considerable amount of time - dropping from $O(N^3)$ to $O(N^2)$.

The new covariance matrix is

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{ss} & \mathbf{K}_{sn} \\ \mathbf{K}_{ns} & \mathbf{K}_{nn} \end{bmatrix}$$

where $\mathbf{K}_{ss}$ is the matrix associated with the old data set, whose Cholesky factor $\mathbf{L}_s$ is known. Matrices $\mathbf{K}_{ns} = \mathbf{K}_{sn}^T$ and $\mathbf{K}_{nn}$ are due to the new samples. The Cholesky factor $\mathbf{L}$ of $\mathbf{K}$ is obtained via the following procedure of partial Cholesky decomposition,

1. Cholesky decomposition: $\mathbf{K}_{nn} = \mathbf{L}_n \mathbf{L}_n^T$.
2. Linear solve: $\mathbf{L}_{sn} = \mathbf{L}_s^{-1} \mathbf{K}_{sn}$
3. Assemble the Cholesky factor,

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_s & \mathbf{O} \\ \mathbf{L}_{sn}^T & \mathbf{L}_n \end{bmatrix}$$

Next, the new $\mathbf{F}$ matrix is

$$\mathbf{F} = \mathbf{L}^{-1}\mathbf{H} = \begin{bmatrix} \mathbf{L}_s & \mathbf{O} \\ \mathbf{L}_{sn}^T & \mathbf{L}_n \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{H}_s \\ \mathbf{H}_n \end{bmatrix} = \begin{bmatrix} \mathbf{F}_s \\ \mathbf{L}_n^{-1}(\mathbf{H}_n - \mathbf{L}_{sn}^T \mathbf{F}_s) \end{bmatrix}$$

where block matrix inversion (https://en.wikipedia.org/wiki/Block_matrix#Block_matrix_inversion) is used and $\mathbf{F}_s = \mathbf{L}_s^{-1}\mathbf{H}_s$ is known from the old model.

Finally, since $\mathbf{F}_s = \mathbf{Q}_s \mathbf{R}_s$ is known, one can utilize the QR update algorithm (http://www.ams.org/journals/mcom/1976-30-136/S0025-5718-1976-0431641-8/) to obtain the QR decomposition of $\mathbf{F}$. Such algorithm has been implemented in standard packages for scientific computation, such as scipy.linalg.qr_insert (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.qr_insert.html).

With the new coefficient matices $\mathbf{L}$, $\mathbf{F}$, $\mathbf{Q}$ and $\mathbf{R}$, the GP model is updated with the new sample data set.

Final note

To be fair, it is worth mentioning the existence of adjoint-based optimization. This methodology applies to optimization problems constrained by partial differential equations (PDEs). One can incorporate the *adjoint* capability in their PDE solver and enable the direct gradient calculation of the expensive objective and constraint functions. This approach applies to optimization problems involving CFD and FE solvers. The adjoint method would be a different (but interesting) story that will be discussed in the next module.