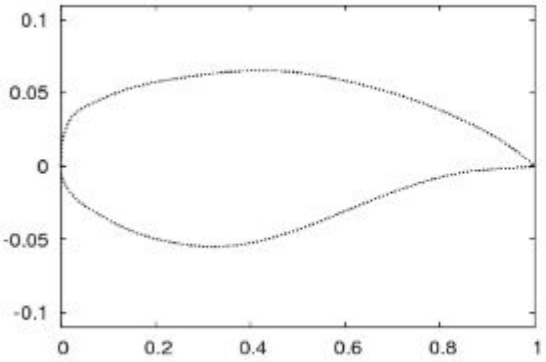
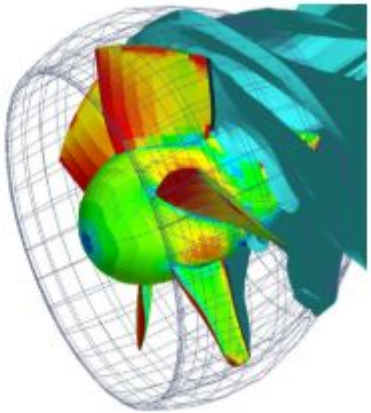
Surrogate modelling and BO introduction

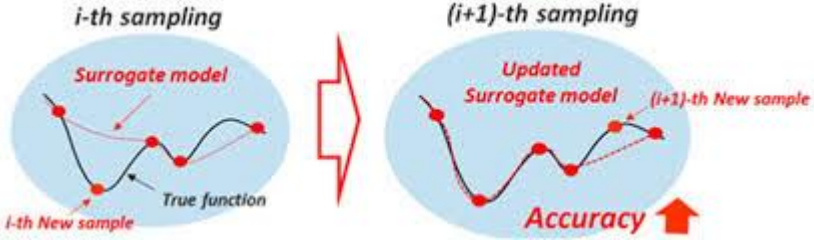
Motivation

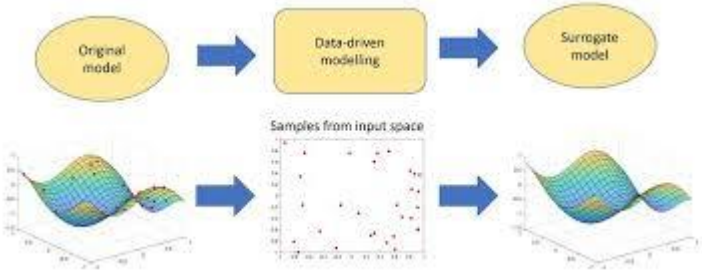
The idea to replace real function evaluations by models that only mimic the true values of the fitness function is rather obvious.

Such expensive fitness function evaluations typically arise in engineering applications. Finite element methods and/or computational fluid dynamics methods are often considered to model a system. Such systems may be, for instance, airfoils or ship propulsion systems


0.1
0.05
0
-0.05
-0.1
0
0.2
0.4
0.6
0.8
1

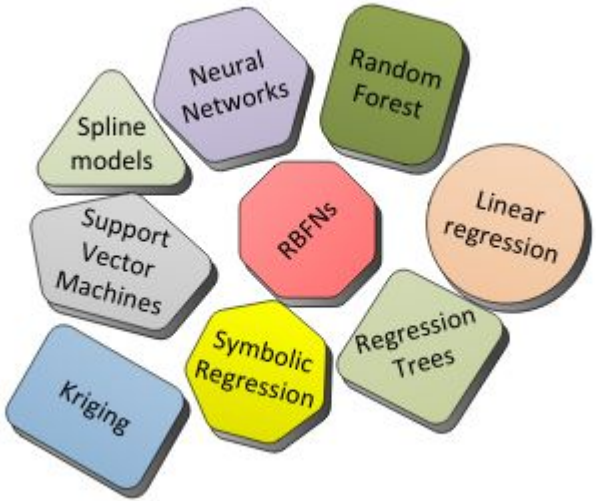



i-th sampling
Surrogate model
True function
i-th New sample
(i+1)-th sampling
Updated
Surrogate model
(i+1)-th New sample
Accuracy


Original model
Data-driven modelling
Surrogate model
Samples from input space

Synonyms

• metamodels, • surrogates, • response surface models, • approximation models, • simulation models, • data-driven models, and • emulators,


Neural Networks
Random Forest
Spline models
Linear regression
RBFNs
Support Vector Machines
Regression Trees
Symbolic Regression
Kriging

Linear models

Linear models consist of a combination of linear predictor functions of each input to model the output. The basic linear model reads

$y = \beta 0 + \beta 1x1 + \beta 2x2 + \cdots + \beta nxn + \varepsilon$

with xi the underlying data to be modelled,

βi being corresponding coefficients that need to be adjusted to fit the model to the data, and ε being an error term.

A polynomial model, for example, takesthe following form

$y = a0 + a1x + a2x2 + a3x3 + \cdots + anxn + \varepsilon.$

Decision trees and random forests

Decision trees model the objective function by using tree-based approximations. At each node of the tree a split is made on the basis of a decision variable value. The prediction of a new point is given by the mean value of associated points.

In random forest regression, a large number of decision trees is combined into an ensemble predictor. Usually, each tree in the ensemble is fitted using a subset of evaluated points to avoid overfitting (bagging). Predictions of new individuals are then given by a cumulated mean of all predictors in the ensemble

Decision trees and random forests

The benefit of decision trees is a good interpretability of the received data structures. Decision trees, as well as random forests, are rather fast in comparison to other, more sophisticated models and are able to handle different types of variables, even binary and integer ones.

In contrast to decision trees, random forests are more complex, and thus harder to analyse and interpret. In return, decision trees may provide poorer fits when applied to more complex functions. Both methods do not provide smooth surfaces and have a risk of overfitting if the trees are allowed to grow too large.

Artificial neural networks and deep learning

Neural networks are methods from the field of computational intelligence mimicking natural systems and processes. They are inspired by the brain of creatures which utilise so-called connected neurons to learn and approximate the behaviour of a function. Neurons are weighted transform functions and the ability to model complex structures is received by a certain amount of connectivity

Complex structured networks with multiple processing layers and/or multiple non-linear transformations and stacked model approaches are also called deep learning approaches. These produce excellent results in approximation and especially classification tasks. However,while providing rather accurate results, deep learning is computational highly complex and needs a lot of computational resources.

Symbolic regression

Symbolic regression [9] is a high-level method to fit a human-readable mathematical model. It

is based on genetic programming, and thus evolves a model using an evolutionary population based approach. The individuals are trees for building high-level expressions based on mathematical operators (+,−,sin, cos, exp...) and the input data.

Although such methods feature a high computational demand in building the model, the prediction afterwards, depending on the complexity of the model built, can be rather fast. Moreover, the resulting models can be analysed and interpreted easily.

Genetic algorithms

They are commonly used to generate high-quality solutions for optimization problems and search problems. Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation.

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from “open parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Fitness Score

A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.

Operators of Genetic Algorithms

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

Particle swarms optimization

Particle Swarm Optimization was proposed by Kennedy and Eberhart in 1995. As mentioned in the original paper, sociobiologists believe a school of fish or a flock of birds that moves in a group “can profit from the experience of all other members”. In other words, while a bird flying and searching randomly for food, for instance, all birds in the flock can share their discovery and help the entire flock get the best hunt.

Utilized for minimum cost functions |

Similar to the flock of birds looking for food, we start with a number of random points on the plane (call them **particles**) and let them look for the minimum point in random directions. At each step, every particle should search around the minimum point it ever found as well as around the minimum point found by the entire swarm of particles. After certain iterations, we consider the minimum point of the function as the minimum point ever explored by this swarm of particles.

Kriging or Gaussian process regression

Kriging or Gaussian process regression [28] is a method to model values by Gaussian processes. It originates from geostatistics and is used to model the error term of the model instead of the linear coefficients. Its simplest form can be written as

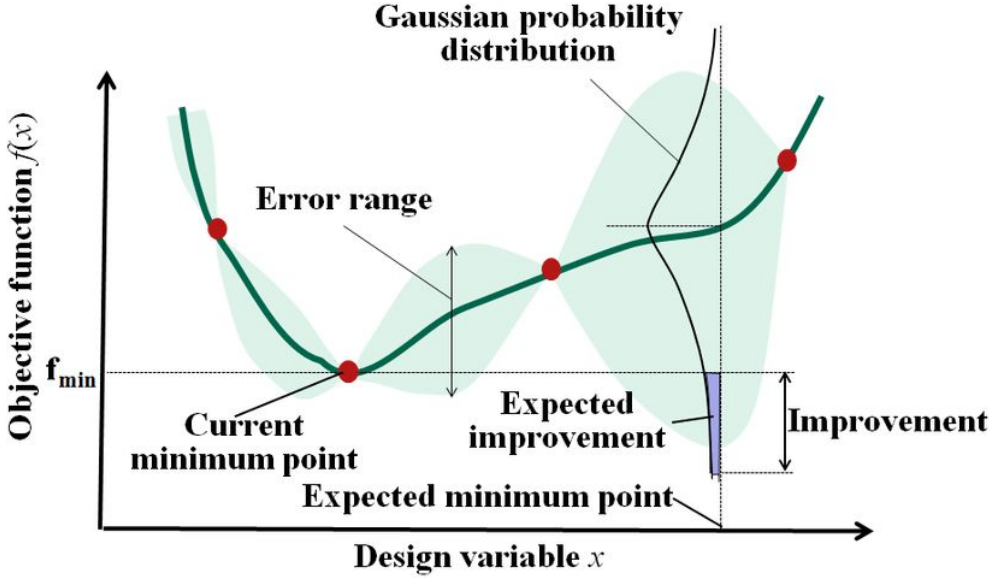
$$y = \beta 0 + \varepsilon$$

with $\beta 0$ being the mean of the observed/considered values and ε is expressed by an Gaussian stochastic process. This error term ε is modelled with the help of a covariance distance matrix. The correlation between errors is related to the distance between the corresponding points and the covariance matrix is utilised to predict unknown candidates.

The outstanding feature of Kriging models is that they provide an uncertainty measure for the prediction.

Another advantage is their suitability for complex functions. However, due to the

computational effort needed, Kriging models are not suitable for high dimensional data. One term closely connected to Kriging modelling is expected improvement.



Bayesian Optimization

Arun Kumar Rajasekaran

BO

Many engineering, scientific, and industrial applications including automated machine learning (e.g., hyper-parameter tuning) involve making design choices to optimize one or more expensive to evaluate objectives.

Some examples include tuning the knobs of a compiler to optimize performance and efficiency of a set of software programs; designing new materials to optimize strength, elasticity, and durability; and designing hardware to optimize performance, power, and area.

Bayesian Optimization (BO) is an effective framework to solve black-box optimization problems with expensive function evaluations. The key idea behind BO is to build a cheap surrogate model (e.g., Gaussian Process) using the real experimental data; and employ it to intelligently select the sequence of function evaluations using an acquisition function, e.g., expected improvement (EI).

When to use BO

- The analytical expression for f(x) is unknown (it is a black box). In this situation, we cannot use other techniques like finding the optimal solution analytically because we don’t have the expression to work with.

- The cost of finding f(x) for a given value of x is very high. This is often true in experimental settings, where f(x) refers to a physical process that we are studying, or when we are trying to optimize the hyperparameters of a neural network, and each time we choose a set of hyperparameters we have to completely re-train the network. If this was not true, we could use simpler approaches like grid search to find the optimal value of x, because each evaluation of f(x) would be cheap.

An intuitive explanation

Bayesian optimization uses two functions to guide the optimization process: a surrogate function and an acquisition function. The surrogate function, $g(\cdot)$, addresses the problem described above where we have no analytical expression for f(x) - the surrogate function acts as our best guess of the form of f(x) given the current information.

The acquisition function, $u(\cdot)$, guides our choice of the next sample of x to sample from the objective function. The acquisition function must balance exploration and exploitation to find the globally optimal value of x as quickly as possible.

An intuitive explanation

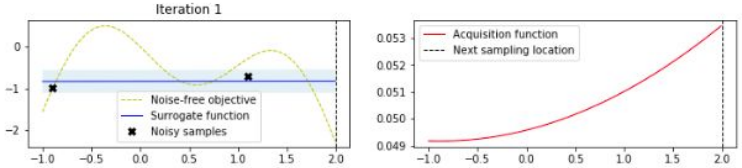
- Iterate for t=1,2,…,T steps for sampled points, (x,y), that are added to the set D1:t−1.

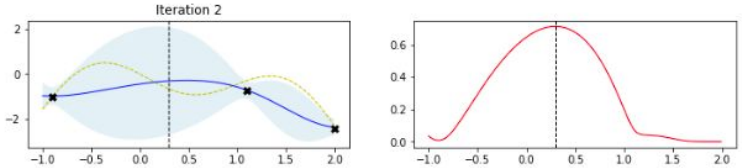
- Select the next point to sample, xt, by finding the argmax of the acquisition function, i.e.:

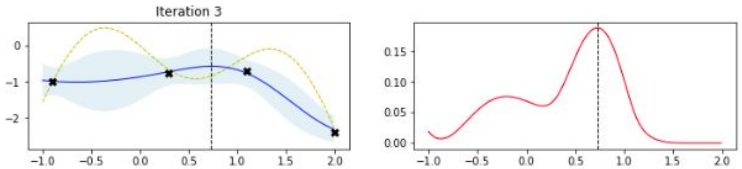
  $$x_t = \operatorname{argmax}_{x \in u} u(x \mid D_{1:t-1})$$

- Sample the objective function at this point: yt=f(xt). Add this sample to the set, D1:t={D1:t−1,(xt,yt)}

- Update the surrogate function, $g(\cdot)$ with the newly sampled point (xt,yt)


Iteration 1
Noise-free objective
Surrogate function
Noisy samples
Acquisition function
Next sampling location

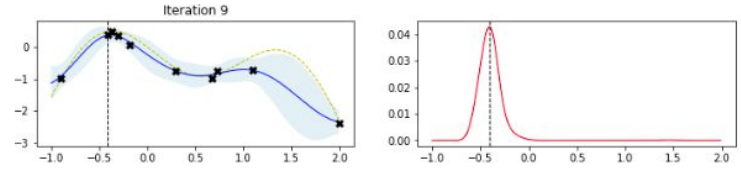

Iteration 2

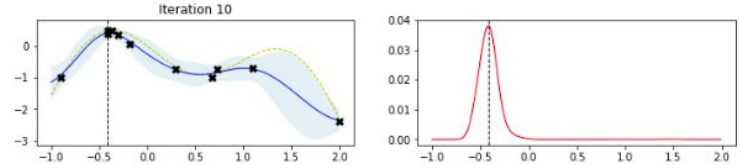

Iteration 3

▪

▪

▪


Iteration 9


Iteration 10

One more look into the optimization process

1. Define the black box function $f(x)$, the acquisition function $a(x)$ and the search space of the parameter x.
2. Generate *some initial values of* x randomly, and measure the corresponding outputs from $f(x)$.
3. Fit a Gaussian process model $m(X, y)$ onto $X = x$ and $y = f(x)$. In other words, $m(X, y)$ serves as a surrogate model for $f(x)$!
4. The acquisition function $a(x)$ then uses $m(X, y)$ to generate new values of x as follows. Use $m(X, y)$ to predict how $f(x)$ varies with x. The value of x which leads to the largest predicted value in $m(X, y)$ is then suggested as the next sample of x to evaluate with $f(x)$.
5. Repeat the optimization process in steps 3 and 4 until we finally get a value of x that leads to the global optimum of $f(x)$. Note that all historical values of x and $f(x)$ should be used to train the Gaussian process model $m(X, y)$ in the next iteration — as the number of data points increases, $m(X, y)$ becomes better at predicting the optimum of $f(x)$.

L-BFGS

“Basically think of L-BFGS as a way of finding a (local) minimum of an objective function, making use of objective function values and the gradient of the objective function. That level of description covers many optimization methods in addition to L-BFGS though.”

First order method means gradients (first derivatives) (and maybe objective function values) are used, but not Hessian (second derivatives). Think of, for instance, gradient descent and steepest descent, among many others.

Second order method means gradients and Hessian are used (and maybe objective function values). Second order methods can be either based on

1. “Exact" Hessian matrix (or finite differences of gradients), in which case they are known as Newton methods

2. Quasi-Newton methods, which approximate the Hessian based on differences of gradients over several iterations, by imposing a "secant" (Quasi-Newton) condition. There are many different Quasi-Newton methods, which estimate the Hessian in different ways. One of the most popular is BFGS. The BFGS Hessian approximation can either be based on the full history of gradients, in which case it is referred to as BFGS, or it can be based only on the most recent m gradients, in which case it is known as limited memory BFGS, abbreviated as L-BFGS. The advantage of L-BFGS is that is requires only retaining the most recent m gradients, where m is usually around 5 to 20, which is a much smaller storage requirement than n*(n+1)/2 elements required to store the full (triangle) of a Hessian estimate, as is required with BFGS, where n is the problem dimension. Unlike (full) BFGS, the estimate of the Hessian is never explicitly formed or stored in L-BFGS (although some implementations of BFGS only form and update the Choelsky factor of the Hessian approximation, rather than the Hessian approximation itself); rather, the calculations which would be required with the estimate of the Hessian are accomplished without explicitly forming it. L-BFGS is used instead of BFGS for very large problems (when n is very large), but might not perform as well as BFGS. Therefore, BFGS is preferred over L-BFGS when the memory requirements of BFGS can be met. On the other hand, L-BFGS may not be much worse in performance than BFGS.

https://medium.com/@okanyenigun/step-by-step-guide-to-bayesian-optimization-a-python-based-approach-3558985c6818