# `CLoVER` software package Documentation

Alexandre Marques, Remi Lam, Karen Willcox

May 11, 2019

## Contents

## 1 Introduction

CLoVER (**C**ontour **Lo**cation **V**ia **E**ntropy **R**eduction) is an active learning algorithm that leverages multiple information sources to locate the zero contour of a function. CLoVER was introduced in the paper

A.N. Marques, R.R. Lam, and K.E. Willcox, Contour location via entropy reduction leveraging multiple information sources, Advances in Neural Information Processing Systems 31, 2018, pp. 5222-5232.

This document describes the usage and organization of `CLoVER.py`, a software written in `Python 2.7.12` that implements the CLoVER algorithm.

Examples of applications of this software package are found in the "applications" folder.

## 2 Installation

To execute `CLoVER`, first install the following `Python` packages:

- `numpy` (`www.numpy.org`)
- `scipy` (`www.scipy.org`)
- `nlopt` (`https://nlopt.readthedocs.io/en/latest/`)

To install `CLoVER`, clone or download the latest release from the following git repository.

`https://github.com/anmarques/CLoVER`

The `Python` module `CLoVER.py` is located within the "source" folder. To execute `CLoVER`, one must import the module `CLoVER.py` as follows (within `Python` environment):

```
>>> import sys
>>> sys.path.append(<path to CLoVER.py>)
>>> import CLoVER
```

## 3 Executing CLoVER

The main function within module `CLoVER.py` is also called `CLoVER`. The syntax of the function `CLoVER` is the following (asterix denotes required input):

```
>>> import CLoVER
>>> f, history = CLoVER.CLoVER(*g, *f, *cost, *noiseVariance, *samplePoints, *nIter, tolH,
tolX, epsilon, integrationPoints, integrationWeights, log)
```

The inputs to and outputs from function `CLoVER` are described below.

### Notation

- $n_{IS}$ = number of information sources.
- $d$ = dimension of parameter space.
- $x$ = set of points in parameter space, represented as a `numpy` array of size $d \times n$, where $n$ is the number of points.
- IS$\ell$ = information source $\ell$.

### Inputs

- `g`: list with $n_{IS}$ functions. Each entry of the list is a function corresponding to an information source. The syntax used to define information sources is discussed in Sect. 4.
- `f`: object of the MISGP class. The MISGP class defines multi-information-source Gaussian process surrogates. The input to `CLoVER` reflects the prior surrogate model used to initialize the iterative process. Further information about the MISGP class is found in Sect. 5.
- `cost`: list with $n_{IS}$ floats or functions. Each entry of the list corresponds to the cost of querying the corresponding information source. If entry is a float, the cost is assumed constant. If entry is a function $c$, then cost = $c(x)$.
- `noiseVariance`: list with $n_{IS}$ floats or functions. Each entry of the list corresponds to the variance of the Gaussian noise associated with the corresponding information source. If entry is a float, the variance is assumed constant. If entry is a function $\lambda$, then variance = $\lambda(x)$.

- `samplePoints`: $d \times n_s$ `numpy` array, where $n_s$ = number of points used as candidates for sampling in parameter space.
- `nIter`: integer. Maximum number of iterations.
- `tolH`: float. Stopping criterion. `CLoVER` stops if contour entropy smaller than `tolH`. The default value is $1 \times 10^{-4}$.
- `tolX`: $d \times 1$ `numpy` array. Maximum distance (in each coordinate direction) between samples. The default value is $d * [1 \times 10^{-10}]$.
- `epsilon`: float or function. $\epsilon$ parameter used in the definition of contour entropy. If float, then $\epsilon$ = `epsilon`$*\sigma(0, x)$, where $\sigma$ is the standard deviation of the posterior surrogate model. If function handle $\varepsilon$, then $\epsilon = \varepsilon(f, x)$. The function $\varepsilon$ must be accept two arguments. $f$: an object of the MISGP class (current surrogate model), and $x$: set of points in parameter space. The default value is 2.0.
- `integrationPoints`: integer or `numpy` array. Integration points used to compute contour entropy. If integer, it defines the number of integration points. Integration is performed using the importance sampling strategy described in Chevalier et. al (Technometrics, 2014). If `numpy` array, it defines numerical quadrature points. The input must be of size $d \times n_q$, where $n_q$ = number of quadrature points. The corresponding quadrature weight are defined in `integrationWeights`. The default value is 2500.
- `integrationPoints`: `None` or `numpy` array. Quadrature weights used to compute contour entropy. If `integrationPoints` is integer, then this input is ignored. Otherwise, `integrationPoints` must be a `numpy` with $n_q$ entries, where $n_q$ = number of quadrature points. The default value is `None`.
- `log`: boolean. If `True` a log file named `CLoVER_history.txt` is updated at every iteration. The log file contain the same information as the `history` output described below. The default value is `False`.

## Outputs

- `f`: object of the MISGP class. Posterior multi-information-source Gaussian process surrogate model at the end of execution.
- `history`: `numpy` array of size $n_q \; times(5 + d + n_h)$, where $n_q$ = total number of queries, and $n_h$ = total number of hyperparameters (including all mean functions and covariance kernels). Each row corresponds to a query to one of the information sources. The columns contain the following information

  1. iteration number.
  2. contour entropy (after the current query).
  3. total cost (after the current query).
  4. information source queried.
  5. point queried ($d$ entries).
  6. observation.
  7. hyperparameters ($n_h$ entries).

  The hyperparamters are displayed in the following order:

  - hyperparameters to mean function of IS0 (if any).
  - hyperparameters to mean function of the discrepancy between IS0 and IS1 (if any).

    $\vdots$

  - hyperparameters to mean function of the discrepancy between IS0 and IS$n_{IS}$ (if any).
  - hyperparameters to covariance kernel of IS0 (if any).
  - hyperparameters to covariance kernel of the discrepancy between IS0 and IS1 (if any).

    $\vdots$

  - hyperparameters to covariance kernel of the discrepancy between IS0 and IS$n_{IS}$ (if any).

## 4 Information sources

Information sources are represented by python functions that accept one argument and return two outputs. The argument is a $d \times n$ `numpy` array, where $d$ = dimension of parameter space, and $n$ = number of points being queried. The outputs are:

1. `numpy` array with $n$ entries corresponding to observations of the information source.
2. float corresponding to the cost (runtime) of making observations.

## 5 Gaussian Process surrogates

This release of CLoVER includes an implementation of the multi-information source Gaussian process (MISGP) surrogate formulation introduced by Poloczek et al. (NeurIPS, 2017). This implementation is reflected in libraries `GP.py` and `MISGP.py`. However, CLoVER supports the use of alternate implementations as long as they provide a class of MISGP objects with the following parameters and methods.

### Notation

- $n_{IS}$ = number of information sources.
- $d$ = dimension of parameter space.
- $n_t$ = number of design points used to train the surrogate model.

### Parameters

- `nIS`: integer. Number of information sources.
- `source`: integer or list of integers. Information sources corresponding to the design points used to train the MISGP object.
- $x$: `numpy` array of size $d \times n_t$ Design points used to train MISGP object.
- $y$: `numpy` array of size $n_t$. Observations at design points used to train MISGP object.

## Methods

- `dimension()`: returns the dimension of parameter space
  - **inputs:** none.
  - **output:** integer. Dimension of parameter. space
- `applyKinverse(M)`: computes $K^{-1}M$, where $K$ is the covariance matrix between all training data of the MISGP object.
  - **input:** `numpy` array of size $n_t \times n_e$, where $n_e \geq 1$ is any positive integer.
  - **output:** `numpy` array of size $n_t \times n_e$ corresponding to $K^{-1}M$.
- `evaluateCovariancePrior(source`$_1$`,` $x_1$`, source`$_2$`,` $x_2$`)`: evaluates the prior covariance kernel between set 1 of points (source$_1$, $x_1$) and set 2 of points (source$_2$, $x_2$).
  - **inputs:**
    1. `source`$_1$: integer or list of integers. Information sources of design points $x_1$.
    2. $x_1$: $d \times n_1$ `numpy` array, where $n_1$ = number of points in set 1. Design points of set 1.
    3. `source`$_2$: integer or list of integers. Information sources of design points $x_2$.
    4. $x_2$: $d \times n_2$ `numpy` array, where $n_2$ = number of points in set 2. Design points of set 2.
  - **output:** `numpy` array of size $n_1 \times n_2$. Prior covariance between set 1 and set 2.
- `evaluateMeanPrior(source,` $x$`)`: evaluates the prior mean function at (source, $x$).
  - **inputs:**
    1. `source`: integer or list of integers. Information sources of design points $x$.
    2. $x$: $d \times n$ `numpy` array, where $n$ = number of design points. Design points at which the mean function is evaluated.
  - **output:** `numpy` array of size $n$. Prior mean function at (source, $x$).
- `evaluateVariancePrior(source,` $x$`)`: evaluates the prior variance at points (source, $x$).
  - **inputs:**
    1. `source`: integer or list of integers. Information sources of design points $x$.
    2. $x$: $d \times n$ `numpy` array, where $n$ = number of design points. Design points at which variance is evaluated.
  - **output:** `numpy` array of size $n$. Prior variance at (source, $x$).
- `getHyperparameter()`: returns the hyperparameters of the mean functions and covariance kernels of the MISGP object.
  - **inputs:** none.
  - **output:** tuple with 2 elements. Each element is a list with $n_{IS}$ entries. The first tuple element corresponds to the hyperparameters of the mean functions, whereas the second element corresponds to the hyperparameters of the covariance kernels. Each entry of the list is either a `numpy` array with the values of the hyperparameters, or `None` if the corresponding mean function or covariance kernel has no hyperparameters.

- `getRankTolerance()`: returns the rank tolerance used in the decomposition of the covariance matrix between design points used to train the MISGP object.
  - **inputs:** none.
  - **output:** float. Rank tolerance.
- `hyperparameterDimension()`: returns the number of hyperparameters of the mean functions and covariance kernels of the MISGP object.
  - **inputs:** none.
  - **output:** tuple with 2 elements. Each element is a list with $n_{IS}$ entries. The first list contains the number of hyperparameters of the mean functions, whereas the second list contains the number of hyperparameters of the covariance kernels.
- `setHyperparameter(pm, pk)`: sets new values for the hyperparameters of the MISGP object.
  - **inputs:**
    1. `pm`: list of `numpy` arrays. The list has as many entries as there are mean functions with non-zero number of hyperparameters. Hyperparameters of mean functions.
    2. `pk`: list of `numpy` arrays. The list has as many entries as there are covariance kernels with non-zero number of hyperparameters. Hyperparameters of covariance kernels.
  - **output:** none.
- `update(source, x, y)`: updates the MISGP object with new data.
  - **inputs:**
    1. `source`: integer or list of integers. Information sources corresponding to data points added to MISGP object.
    2. `x`: `numpy` array of size $d \times n$, where $n =$ number of additional data points. Design points to be added to MISGP object.
    3. `y`: `numpy` array of size $n$, where $n =$ number of additional data points. Observations collected at design points $x$.
  - **output:** none.