Welcome to SAT User Guide!

> SAT has not been publicly released yet.
>
> > Contact to me: fov223@gmail.com

Surrogate-based Auto Tuning (SAT) is a MATLAB package for stochastical optimization, SAT includes Kriging (Gaussian Process) modeling method and useful function in SAT framework.

You can refer to **demo.m** (in package's root folder) for usage of SAT intuitively.

SAT Framework

```
Initialization;
while stopping criteria is not satisfied:

    Modeling;
    Infilling;

endwhile
```

Stochastical Modeling Method

Target to different objective problems, SAT provided several types of Kriging as following:

- Ordinary Kriging
- Co-Kriging
- Derivative-enhanced Kriging (coming soon)

Data Arrangement

In SAT, we require user to give data a specified form which is readable by SAT machine.

Training Set

We require each sample and its corresponding value are row-wisely permuted.

```
samples = [sample1; sample2; sample3];
values  = [value1 ; value2 ; value3 ];
```

Commonly Used Variables

Variable's names in demo.m are easy to understand what it means.

Variable name	Description
numDim	Dimension number of objective function
numInitial	Number of initial samples
samples	Coordinates of current training set

Initialization

Problem Definition

Constant Variables

- numDim
- Stopping Criteria:
    - maxIteration
    - maxTime

Function Evaluation

In optimization procedure, it always asks to evaluate function value at a series of points repeatly, in this User Guide we assume objective

function called **YourFunc**, this function outputs a real value.

```
value = YourFunc(sample);
```

Search Region

Our model can predict everywhere in space same as samples, but to predict the value at some point far far away from training data is meaningfulless, since training set has tiny even no influence to that points.

You are required to set a rectangular region **window** where we select sample at every iteration,

```
window = [x_lb, x_ub, y_lb, y_ub]

% x_lb: lower bound in x_axis
% x_ub: upper bound in x_axis
% y_lb: lower bound in y_axis
% y_ub: upper bound in y_axis
```

> Here we assume numDim is 2, then we define window as a rectangular region, window will be a 1-by-2*2 (1-by-2*numDim) vector.

Observation Grid

Function **grid_cut** list the coordinates of discrete points in **window**, we call this set of points observation set (**G**), and this function allocates vectors for prediction (**P**), mean square error (**MSE**) and expected improvement (**EI**) corresponding to points in observation set.

```
[gridLen, G, P, MSE, EI] = grid_cut(window, gridsize);
```

- Input
  - **window** : [lb1, ub1, lb2, ub2, ..., lbk, ubk]
  - **gridsize**: [g1, g2, ..., gk]
- Output
  - **gridLen** : Integer number, number of observation points in grid
  - **G** : gridLen-by-k matrix, list every observation point in every single row
  - **P** : gridLen-by-1 vector, list for fill with prediction value corresponding to every observation point
  - **MSE** : gridLen-by-1 vector, list for fill with mean square error corresponding to every observation point
  - **EI** : gridLen-by-1 vector, list for fill with expected improvement corresponding to every observation point

Variable **gridsize** means how dense observation set to list, it is a 1-by-numDim vector, each component represent how many grid to cut in each axis.

Latin Hypercube Design

Latin Hypercube Design is a space-filling algorithm,

```
samples = LHD(numInitial, numDim, window);
```

Training Set Initialization

Constructing Kriging model needs a set of training data.

Ordinary Kriging

```
values = zeros(size(samples, 1), 1);
for idx = 1:size(samples, 1)
    values(idx) = YourFunc(samples(idx,:));
end
```

Co-Kriging

In Co-Kriging modeling method, it's supposed that there are available to use two fidelities of evaluation function, they are supposed to solve the same problem but have different levels of approximation.

One of them has more accurate, Co-Kriging will construct surrogate for high-fidelity objective.

Modeling

Ordinary Kriging

Construct Kriging model,

```
model = Kriging_info(samples, values);
```

The Kriging kernel function output **model**, that is the handle of current model.

Prediction,

```
for i = 1:gridLen
    [P(i), MSE(i)] = Kriging_pred(G(i, :), model);
    EI(i) = eiMaximum(P(i), values, MSE(i));
    % EI(i) = eiMinimum(P(i), values, MSE(i));
end
```

Co-Kriging

Suppose there are two sets of data:

- High fidelity data: **samples_high** and **values_high**.
- Low fidelity data : **sample_low** and **values_low**.

Construct Co-Kriging model,

```
model = coKriging_info(samples_high, samples_low, values_high, values_low);
```

Prediction,

```
for i = 1:gridLen
    [P(i), MSE(i)] = coKriging_pred(G(i, :), model);
    EI(i) = eiMaximum(P(i), values, MSE(i));
    % EI(i) = eiMinimum(P(i), values, MSE(i));
end
```

Uncertainty Measurement

Mean Square Error (MSE)

Kriging has some useful stochastical properties, it has been developed to have ability to measure the uncertainty of surrogate,

Expected Improvement (EI)

Target to

- Maximization Problem, use **eiMaximum**.
- Minimization Problem, use **eiMinimum**.

Infilling

Choose a new sample

If your object problem is a optimization,

```
[~, new_idx] = max(EI);
```

If your goal is to make model more fitting,

```
[~, new_idx] = max(MSE);
```

Evaluate

```
new_sample = G(new_idx, :);
new_value = YourFunc(new_sample);
```

Append

```
samples(end+1, :) = new_sample;
values(end+1, :) = new_value;
```

"Zoom in" Strategy

Brief Idea

To find the optimal solution, shrink the search region might accelerate optimization procedure.

Shrink Search Region and Create New Window

Function **zoom_shrink_Window** create a info of new window, the new window has the center at optimum sample in training set

```
window_new = zoom_shrink_Window(samples, values, window_old, shrink_ratio, opt_prob)
```

Drop Samples Not Belong to New Window

```
[samples, values] = zoom_shrink_dropoutside(window, samples, values);
```

Disadvantage

On the other hand, this strategy might converge to local optimum

Table of Contents