SAT User Guide Documentation

Release 0.8

NTU COLA Lab

August 27, 2014

1 Introduction **3**
1.1 Preface . . . 3
1.2 SAT Framework . . . 3
1.3 Stochastical Modeling Method . . . 3

2 Data Arrangement **5**
2.1 Training Set . . . 5
2.2 Commonly Used Variables . . . 5

3 Initialization **7**
3.1 Problem Definition . . . 7
3.2 Observation Grid . . . 7
3.3 Latin Hypercube Design . . . 8
3.4 Training Set Initialization . . . 8

4 Modeling **11**
4.1 Kriging . . . 11
4.2 Co-Kriging . . . 11

5 Uncertainty Measurement **13**
5.1 Mean Square Error (MSE) . . . 13
5.2 Expected Improvement (EI) . . . 13

6 Infilling **15**
6.1 New Sampling . . . 15
6.2 Evaluate and Append . . . 15

7 "Zoom in" Strategy **17**
7.1 Brief Idea . . . 17
7.2 Shrink Search Region and Create New Window . . . 17
7.3 Drop Off Samples Not Belong to New Window . . . 17
7.4 Disadvantage . . . 17

8 Indices and tables **19**

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 Preface

Surrogate-Based Auto Tuning (SAT) package is a collection of MATLAB code used for stochastically optimization based on the Kriging modeling method. **demo_kriging.m** and **demo_cokriging.m** can be referred to for an intuitive look in the function of this package.

1.2 SAT Framework

SAT brief framework:

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

    Modeling;
    Infilling;

endwhile
```

1.3 Stochastical Modeling Method

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

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

CHAPTER

TWO

DATA ARRANGEMENT

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

2.1 Training Set

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

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

2.2 Commonly Used Variables

Variable's names in **demo_kriging.m** and **demo_cokriging.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
values	Values of current training set

CHAPTER

THREE

INITIALIZATION

3.1 Problem Definition

3.1.1 Constant Variables

- **numDim**
- **maxIteration**
- **maxTime**

3.1.2 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);
```

3.1.3 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.

3.2 Observation Grid

Given a vector of alternating corresponding lower and upper bounds along with a vector of the corresponding **gridsize**, construct multiple return values for reshaping the structure. It returns a integer number length of the number of observation points in the grid, a length by k matrix called grid containing every observation point in every row, a

length by 1 vector matrix called **P** for containing the prediction value corresponding to ever observation point, a length by 1 vector called **MSE** for containing the mean square error corresponding to every observation point, and a length by 1 vector called **EI** for containing the expected improvement corresponding to every observation point. The output vectors, **P**, **MSE**, and **EI** do not contain actual values from this function:

```
[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.

3.3 Latin Hypercube Design

For **numDim**-dimensional region bounded in **window**, generate a Latin-Hypercube Design. Requires an integer representing the number of designs **numInitial**, an integer representing the dimension of designs **numDim** and a 1 by 2*numDim vector representing the region's edges in each dimension window. Will output a **numInitial** by **numDim** matrix over a region in **window** Initial:

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

3.4 Training Set Initialization

Constructing Kriging model needs a set of training data.

3.4.1 Kriging

Put objective data in values:

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

3.4.2 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.

CHAPTER

FOUR

MODELING

4.1 Kriging

Given **N** data with dimension **numDim**, create a handle of the Kriging model. In this function **samples** is a **n** by **numDim** matrix of sample(feature data) and **values** is an **n** by 1 vector that corresponds with **samples**. Outputs **model** structure:

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

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

Based on the info generated in the function Kriging_info, this function predicts the value y and mean square error mse at the points contained in x:

```
[y,mse] = Kriging_pred(x,Kmodel)
```

In demo script, we predict all points in observation list:

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

4.2 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**.

Given the inputs matrix **samples_high**, **samples_low**, **values_high** and **values_low**, declares the global variables ModelInfo.Xe, ModelInfo.Xc, ModelInfo.ye, and ModelInfo.yc respectively. This function generates and stores the information about the corresponding co-kriging model as a global variable and outputs it:

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

Given the input scalar x and the input structure CKmodel, this function will construct a function f_hat to model the given data from the four matrices inside the Info structure. It will output this function as well as a generated error mse using the mean square error:

```
[f_hat,mse] = coKriging_pred(x, CKmodel)
```

Prediction all points:

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

CHAPTER

FIVE

UNCERTAINTY MEASUREMENT

5.1 Mean Square Error (MSE)

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

5.2 Expected Improvement (EI)

Target to

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

CHAPTER

SIX

INFILLING

6.1 New Sampling

If your object problem is a optimization:

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

If your goal is to make model more fitting:

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

6.2 Evaluate and Append

A stupid method:

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


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

CHAPTER

SEVEN

"ZOOM IN" STRATEGY

7.1 Brief Idea

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

7.2 Shrink Search Region and Create New Window

This function determines a new smaller window and locates the position of edge. The optimal sample in the data set is chosen as the center of the new data. Requires the training set **samples** and **values**, a 1 by 2*numDim vector info of old window **window_old**, a 1 by **numDim** vector specifying the ratios in every dimension **shrink_ratio**, and to set **opt_prob** to be either **max** or **min**. The output is a 1 by 2*numDim vector of the info of the new window:

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

7.3 Drop Off Samples Not Belong to New Window

This function drops off the samples which are outside the window. Requires a 1 by 2*numDim vector info of window region **window**, an N by **numDim** matrix of sample (feature data) **samples**, and an N by 1 vector corresponding to **samples** response **values**. The output **samples_new** is an N by **numDim** matrix with updated samples and values_new is an N by 1 vector of the updated responses:

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

7.4 Disadvantage

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

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*