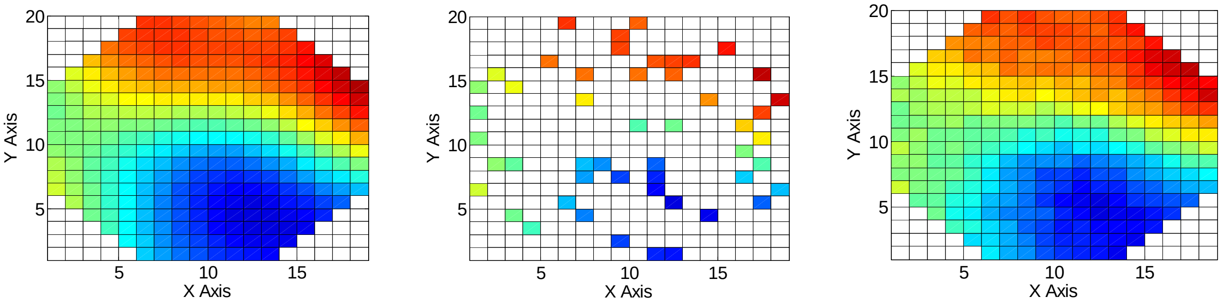
VP: A MATLAB Solver for Spatial Variation Prediction

Wangyang Zhang and Xin Li
ECE Department
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
{wangyan1, xinli}@ece.cmu.edu

Rob Rutenbar
Computer Science Department
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave, Urbana IL 61801
rutenbar@illinois.edu

1. INTRODUCTION

VP is a MATLAB solver for the problem of spatial variation prediction described in [1]. The problem can be illustrated by the example in Fig.1. The spatial variation can be exactly characterized by measuring the performance of interest at all possible sampling locations, as shown in Fig.1 (a). To save the testing cost, it is possible to only measure a small subset of all locations, as shown in Fig.1 (b). Then, VP can be utilized to reconstruct the spatial variation from these partial measurements, and the output is the predicted performance at all locations, as shown in Fig.1 (c).



(a) (b) (c)

Fig.1 An illustration of the spatial variation prediction problem. (a) The true spatial variation obtained by measuring all sampling locations. (b) A subset of the measurements. (c) Reconstructed spatial variation from the measurements in (b).

2. THE VP SOLVER

The VP solver file contained in this package is **solve_vp.m**. The input to **solve_vp.m** is a list of partial measurements like Fig.1 (b), and the output is the reconstructed spatial variation like Fig.1 (c). In the following, we denote the number of measurements as M, the maximum x-coordinate as P and the maximum y-coordinate as Q, which are consistent with the definitions in [1].

2.1 Input Arguments

- *SampleData*:

  A size-M column vector specifying the measured performances
- *SampleX*:

  A size-M column vector specifying x-coordinates of the measurements.
- *SampleY*:

  A size-M column vector specifying y-coordinates of the measurements. The first three input arguments are used to define the partial measurement data. For example, if the 5-th measurement data is take at the location (3, 4) and the performance is 6.2, we will have *SampleData*(5)=6.2, *SampleX*(5)=3, *SampleY*(5)=4.

- *DataFlag*:

  A *P*-by-*Q* matrix specifying the locations of all possible sampling locations. If *DataFlag*(*i*, *j*)=1, it means (*i*, *j*) is a valid measurement location. Invalid locations will have *DataFlag*(*i*, *j*)=0. For example, in the wafer in Fig.1 (a), (5, 2) is a valid measurement location and (5, 1) is invalid. Therefore, we need to set *DataFlag*(5, 2)=1 and *DataFlag*(5, 1)=0 when performing any spatial variation prediction using this dataset.
- *noise* (optional):

  A boolean variable specifying whether measurement data contain significant independent random variation, which appears as noise in spatial variation prediction. The default value of this variable is false, and this option is suggested when it is known that the independent random variation is at least one magnitude less than the spatially correlated variation in the measurement data. When noise is set to true, VP will internally run cross-validation to determine the amount of noise, which may improve the prediction accuracy at the cost of a few hundred times additional computation time.

2.2 Output Arguments

- *PerfEst*:

A *P*-by-*Q* matrix with the recovered spatial variation by VP. *PerfEst* has the same dimension with the input argument *DataFlag*. If *DataFlag*(*i*, *j*)=0, *PerfEst*(*i*, *j*) will be filled with NaN. Otherwise, *PerfEst*(*i*, *j*) is the predicted performance at (*i*, *j*). Note that if *noise* is set to true, for locations that are already measured, the predicted performance is not necessarily equal to the measured performance because the independent variation is automatically subtracted.

3. THE VP TESTER

If measurement data at all locations (e.g. Fig.1 (a)) are known in advance, and the user simply wants to test the accuracy of VP on a known dataset (e.g. compare Fig.1 (c) with Fig.1 (a)), a utility function that helps this process is **test_vp.m**. This function takes all measurement data as an input, samples a subset of these data, performs spatial variation prediction and calculates the prediction error by comparing the prediction with the actual data.

3.1 Input Arguments

- *PerfData*:

  A *P*-by-*Q* matrix specifying the measured performances at all locations. Its format is the same the *PerfEst* defined in Section 2.2: *PerfData*(*i*, *j*)=NaN means (*i*, *j*) is not a valid measurement location, otherwise *PerfData*(*i*, *j*) is the measured performance at location (*i*, *j*).
- *NumExp*:

  The number of measurements used for spatial variation prediction. If the parameters *SampleX* and *SampleY* are not specified, the program will automatically select *NumExp* points from all points in *PerfData* by Latin Hypercube Sampling [2].
- *noise* (optional):

  The same as the noise parameter in Section 2.1.
- *SampleX* (optional):

  The same as the *SampleX* parameter in Section 2.1.
- *SampleY* (optional):

  The same as the *SampleY* parameter in Section 2.1. If *SampleX* and *SampleY* are both specified, they will override the automatic sample generation.

3.2 Output Arguments

- *error*:
  The average error for the prediction calculated by Eq.(18) in [1].
- *PerfEst*:
  The same as the *PerfEst* parameter in Section 2.2.

4. EXAMPLES

Two examples are included in the VP package to illustrate how to use the code. These two examples use a synthetic dataset as shown in Fig.1 (a) to experiment on the two functions in Section 2 and Section 3 respectively.

4.1 Example 1

example1.m illustrates the usage of the **solve_vp** function. In the following we will go through this example in detail.

Example 1 first loads the dataset from **example1.mat**. The variables contained in the dataset include *SampleData*, *SampleX*, *SampleY*, *DataFlag* and *PerfData*. These variables have the same meaning as explained in Section 2-3. A graphical representation of these variables can be found in Fig.1 (a)-(b).

```
load example1;
```

Next, **solve_vp** is called to recover the spatial variation from partial measurements. The recovered spatial variation is stored in *PerfEst*.

```
PerfEst = solve_vp(SampleData,SampleX,SampleY,DataFlag);
```

Finally, we use MATLAB commands to plot Fig.1 (a) and Fig.1 (c) as visual representations of the spatial variation, from *PerfData* and *PerfEst* respectively.

```
TrueData = zeros(19,20);
TrueData(1:18,1:19) = PerfData;
EstData = zeros(19,20);
EstData(1:18,1:19) = PerfEst;

figure(1);
pcolor(TrueData');
xlabel('X Axis');
ylabel('Y Axis');
caxis([4 8]);
title('True wafer map');

figure(2);
pcolor(EstData');
xlabel('X Axis');
ylabel('Y Axis');
caxis([4 8]);
title('Estimated wafer map by 50 samples');
```

4.2 Example 2

example2.m illustrates the usage of the **test_vp** function. In the following we will go through this example in detail.

Example 2 first loads the dataset from **example2.mat**. This dataset only contains one variable, *PerfData*, which is the measured performances at all locations. A graphical representation of it can be found in Fig.1 (a).

```
load example2;
rand('state',1);
```

Next, the average error of spatial variation prediction is tested with different number of sampling points ranging from 20 to 100. Note that it is not necessary to specify the locations of sampling points.

```
for i = 20:10:100
    error(i) = test_vp(PerfData,i);
end
```

In the end, the curve of relative error vs number of samples is plotted.

```
figure(1);
plot(20:10:100,error(20:10:100),'-bo','linewidth',2);
xlabel('Number of Samples');
ylabel('Relative Error');
```

5. REFERENCES

[1] X. Li, R. Rutenbar and R. Blanton, “Virtual probe: a statistically optimal framework for minimum-cost silicon characterization of nanoscale integrated circuits,” IEEE ICCAD, pp. 433-440, 2009.

[2] M. McKay, R. Beckman, and W. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from computer code,” Technometrics, vol. 42, no. 1, pp. 55-61, May. 1979.