State estimation and property-guided exploration for hybrid systems testing

Thao Dang and Noa Shalev,
CNRS/VERIMAG
Grenoble, France

Introduction: Model-Based Testing

- **Hybrid systems**: appropriate high-level model for **analog and mixed-signal circuit**
- **Testing**: suffers less from the 'state explosion' problem and can be applied to real circuits and not only to their models.
- Tester **controls (uncertain) input** and checks whether corresponding observations are **as expected**.
- **Infiniteness** of the state space $\Rightarrow$ notion of **coverage**
- Semi-formal validation

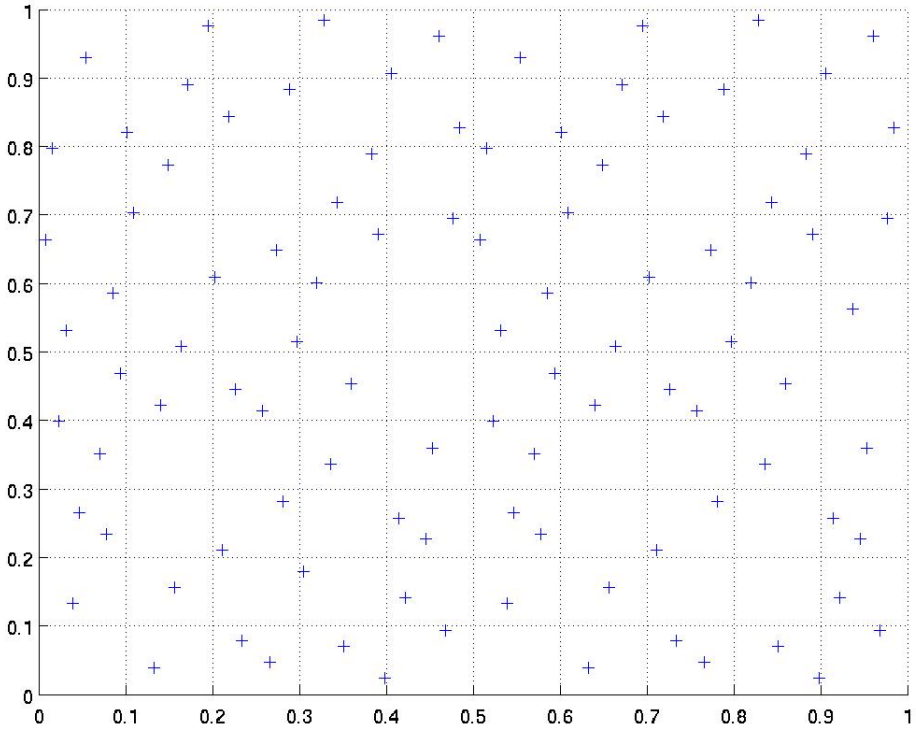
Testing Problems

- A **system under test** (which can be modeled by a hybrid automaton) often operates within some environment.
- The **tester** plays the role of the **environment**. The tester **generates the continuous inputs and control discrete transition** and **observes the output**.
- **Partial observability**: state of the system is not **fully observed** by the tester
- **Testing generation**: automatically generate a set of test cases from the system model to satisfy a coverage criterion.
- **Test execution** under partial observability

Test Coverage

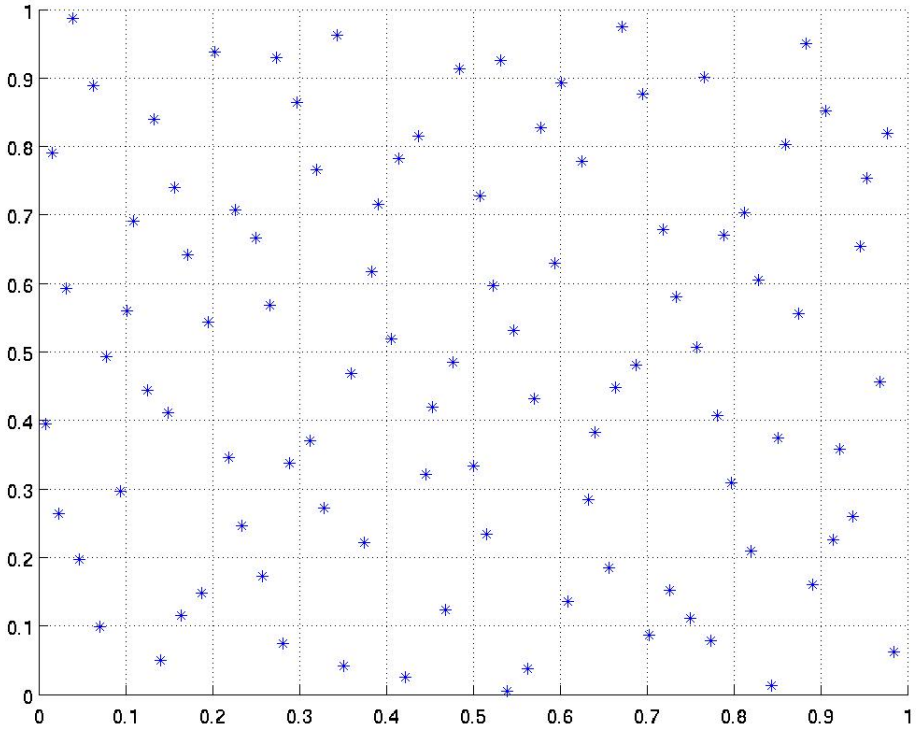
- **Test coverage** is a measure to evaluate testing quality.
- **Star discrepancy** notion in statistics, which characterises the uniformity of the distribution of a point set within a region.
-

Star Discrepancy



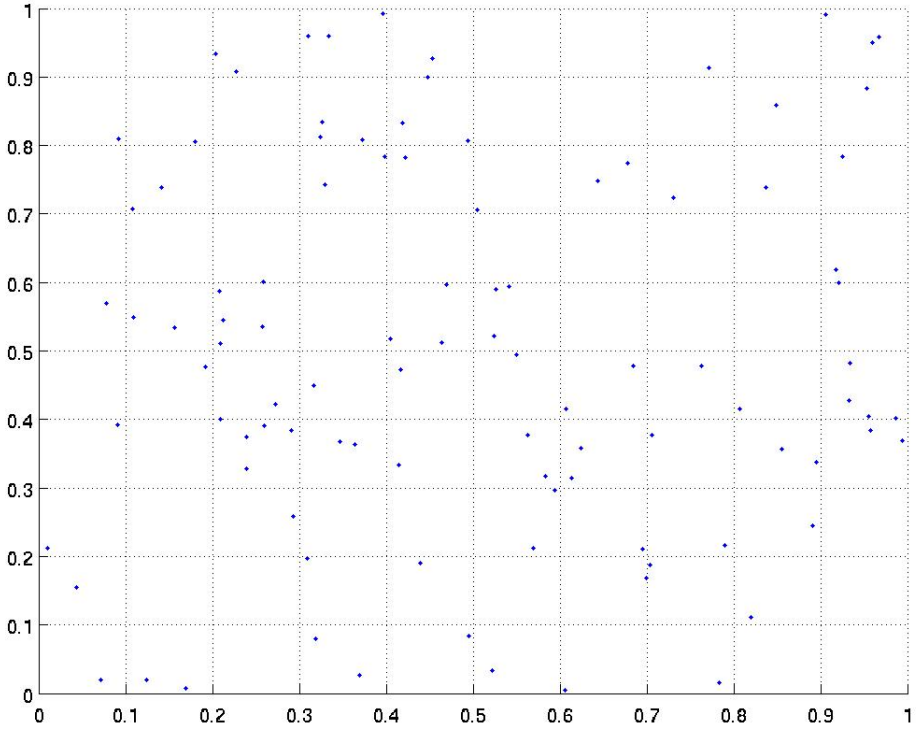
Faure sequence of 100 points. Its star discrepancy value is 0.048.

Star discrepancy



Halton sequence of 100 points. The star discrepancy value is 0.05.

Star Discrepancy



Sequence of 100 points generated by a **pseudo-random function in the C library**. Its star discrepancy value is 0.1.

Test Generation - gRTT (previous work)

- **Randomized** exploration, inspired by probabilistic **motion planning** techniques **RRT** (Random Rapidly-Exploring Trees) in robotics
- **Coverage criteria** reflect testing quality
- **Guided** by coverage criteria

New Results

- Combining **Property-guided** and Coverage-guided
- Test execution under **partial observability**

Combining Property-Guided and Coverage-Guided Sampling

Coverage-Guided

- Bias goal state sampling distribution to improve current coverage
- Good coverage: visited states represent well the reachable set

Property-Guided: We want to bias the exploration towards some regions or some traces.

Combining Property-Guided and Coverage-Guided Sampling

Two steps:

- Property-Guided Sampling to choose a region (using a random walk)
- Coverage-Guided Sampling to choose a goal state within the region

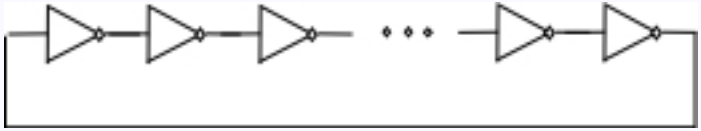
Property-Guided Sampling using a random walk

- Using a **partition** of the continuous state space to construct an **underlying automaton**
- Performing a **random walk** on the automaton to guide the exploration
- Random walk determines the **region** within which the goal state is **sampled**.

Random Walk: Metropolis-Hastings algorithm

- Given an automaton $G(V, E)$, a **target probability distribution** $\pi = \{\pi_v \mid v \in V\}$.
- $$\begin{cases} p_{uv} = \frac{1}{d_u} min\{\frac{d_u \pi_v}{d_v \pi_u}, 1\} & \text{if } v \text{ is adjacent node of } u \\ p_{uv} = 1 - \sum_{w \neq u} p_{uw} & \text{if } v = u \\ p_{uv} = 0 & \text{otherwise} \end{cases}$$
- **Stationary distribution** is the **target distribution** π
- **Hitting times** $\mathcal{O}(rN^2)$ (where N is the number of vertices, $r = \max \pi_u/\pi_v$)

Experimental results: Ring Oscillator



One input variable: source voltage of the circuit, between $1.6V$ and $2.4V$.

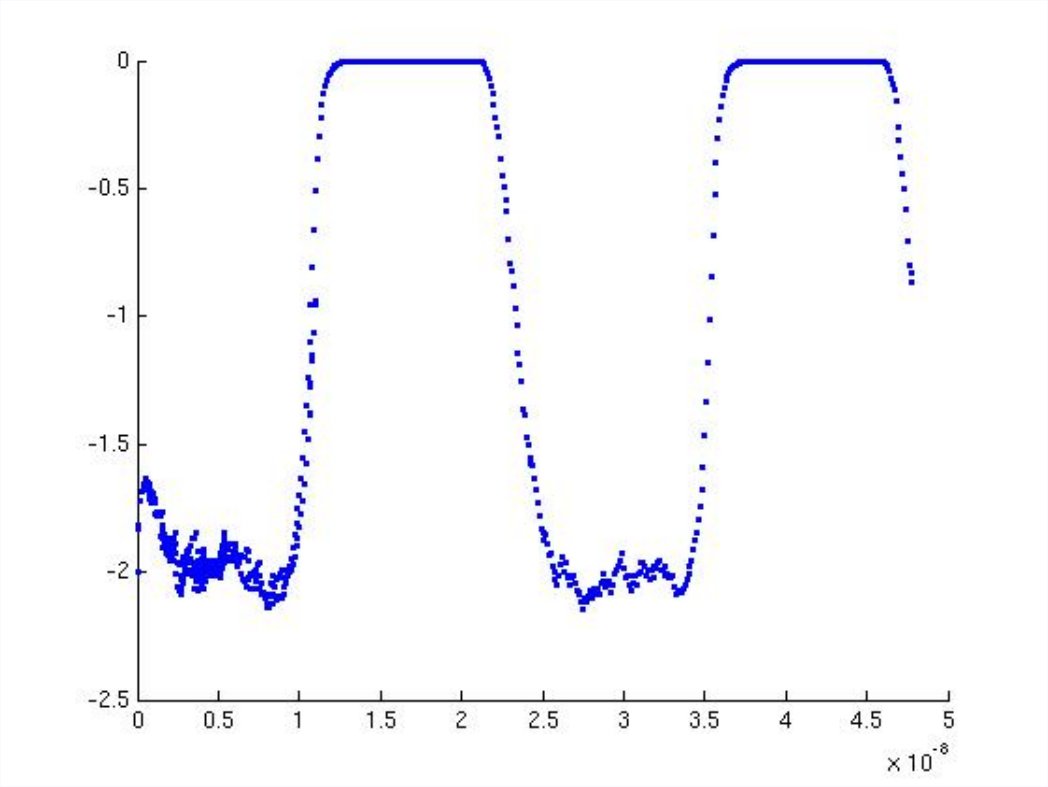
9 state variables: output voltages of each inverter

Partition of the state space corresponding to two "binary" states of the outputs of the inverters:

$$s_1 = (low, high, low, high, low, high, low, high, low)$$

$$s_2 = (high, low, high, low, high, low, high, low, high)$$

Ring Oscillator - Without Random Walk



Maximum value 0, **minimum value** -2.1458

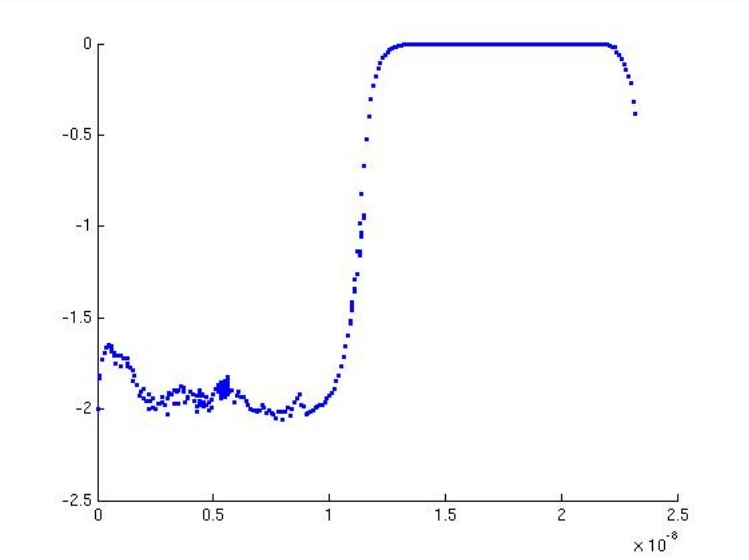
Ring Oscillator - With Random Walk

$$s_1 = (low, high, low, high, low, high, low, high, low)$$

$$s_2 = (high, low, high, low, high, low, high, low, high)$$

Metropolis-Hastings Algorithm $\pi_1 = 0.8$ and $\pi_2 = 0.2$

from $\rightarrow$ to	0	1
0	0.875	0.125
1	0.5	0.5



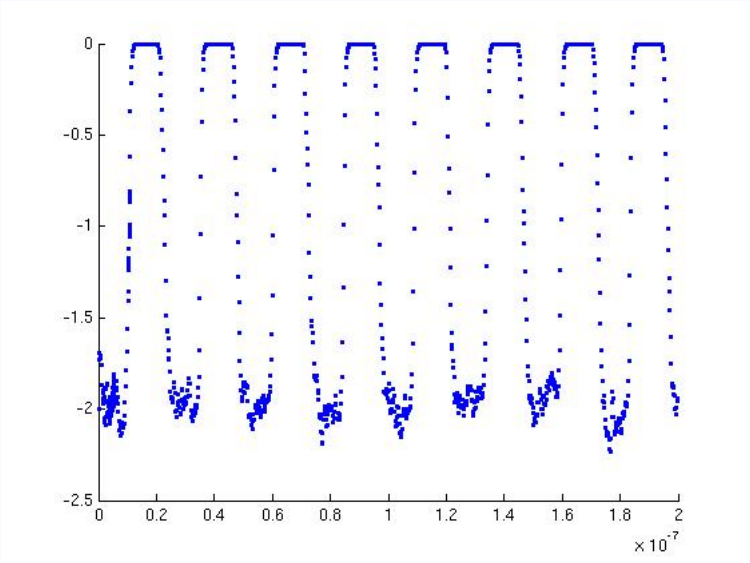
Maximum value 0, **minimum value** -2.1725

Ring Oscillator - Randowm Walk and Favoring Time

$$s_1 = (low, high, low, high, low, high, low, high, low)$$

$$s_2 = (high, low, high, low, high, low, high, low, high)$$

Metropolis-Hastings Algorithm $\pi_1 = 0.8$ and $\pi_2 = 0.2$



Maximum value 0, **minimum value** -2.2413

Partial Observability

- Locations are not observable
- Outputs of sensors are modeled using an observation function:

$$\begin{aligned} x[k+1] &= f(x[k], u[k]) \\ y[k] &= h(x[k]) \end{aligned}$$

Newton Observer: estimate the state $x[k]$ from a sequence of consecutive inputs and corresponding observations

Note: test a model/approximation w.r.t. a set of expected behaviors

Newton Observer

Let $U_{k,k+N-1}$ be a vector of N consecutive inputs that is to be applied to the system at time k:

$$U_{k,k+N-1} = \begin{pmatrix} u[k] \\ u[k+1] \\ \cdots \\ u[k+N-1] \end{pmatrix}$$

Under this continuous input sequence and starting from the state $x[k]$ (that we need to estimate), the system under test produces a vector of observations

$$Y_{k,k+N-1} = \begin{pmatrix} y[k] \\ y[k+1] \\ \cdots \\ y[k+N-1] \end{pmatrix}$$

Newton Observer (2)

We define the following vector of functions:

$$H(x, U_{0,N-1}) = \begin{pmatrix} h(x) \\ h \circ f^{u[0]}(x) \\ \dots \\ h \circ f^{u[N-1]}(x) \circ \dots \circ f^{u[0]}(x) \end{pmatrix}$$

If the system is N-observable (with $N \geq 1$) at state $\tilde{x}$ if there any sequence U of N control inputs such that $\tilde{x}$ is the unique solution of the following equation:

$$H(\tilde{x}, U) \ = H(\xi, U)$$

We want to compute x_{k-N+1} satisfying: $Y_k - H(x_{k-N+1}, U_k) = 0$

Newton iteration to compute $\tilde{x}_{k-N+1}$

$$\alpha^{j+1} = \alpha^j + [\frac{\partial H}{\partial \alpha}(\alpha^j, U_k]^{-1}(Y_k - H(\alpha^j, U_k))$$

Hybrid Observer - Decision Function

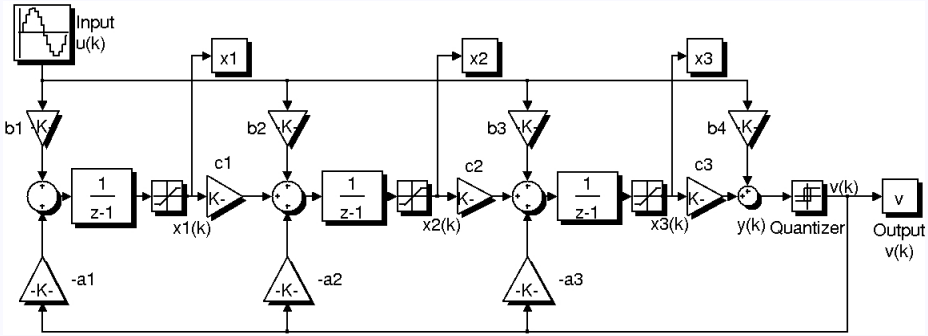
$i = 0$, $S_{init} = \{(q, x_0) \mid q \in Q\}$, $S_n = S_{init} \cup Succ_d(S_{init})$
REPEAT
  $S_i = S_n \cup Succ_d(S_n)$ /* S_i *set of possible current states* */
  $S_n = \emptyset$ /* S_n *set of possible next states* */
  FORALL$((q, x) \in S_i)$
    /* *Choose a feasible input sequence* U *of length* N */
    $U = InputSeq(T, (q, x))$

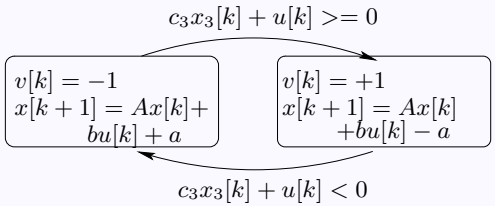
    /* *Apply the input sequence* U *and observe the outputs* */
    $Y = Observation(U)$

    /* *Using Newton's algorithm to estimate the state at time* iN */
    $\xi = Newton(q, U, Y)$

    /* *Adding all the continuous successors to the set* S_n,
    they will be explored in the next iteration */
    IF $((||\xi - x|| \leq \varepsilon))$ $S_n = S_n \cup \{Succ_c((q, x), U)\}$)
  }
  If $(S_n = \emptyset)$ **RETURN 'fail' verdict**
UNTIL $i = i_{max} \vee S_n = \emptyset$

Delta-Sigma circuit

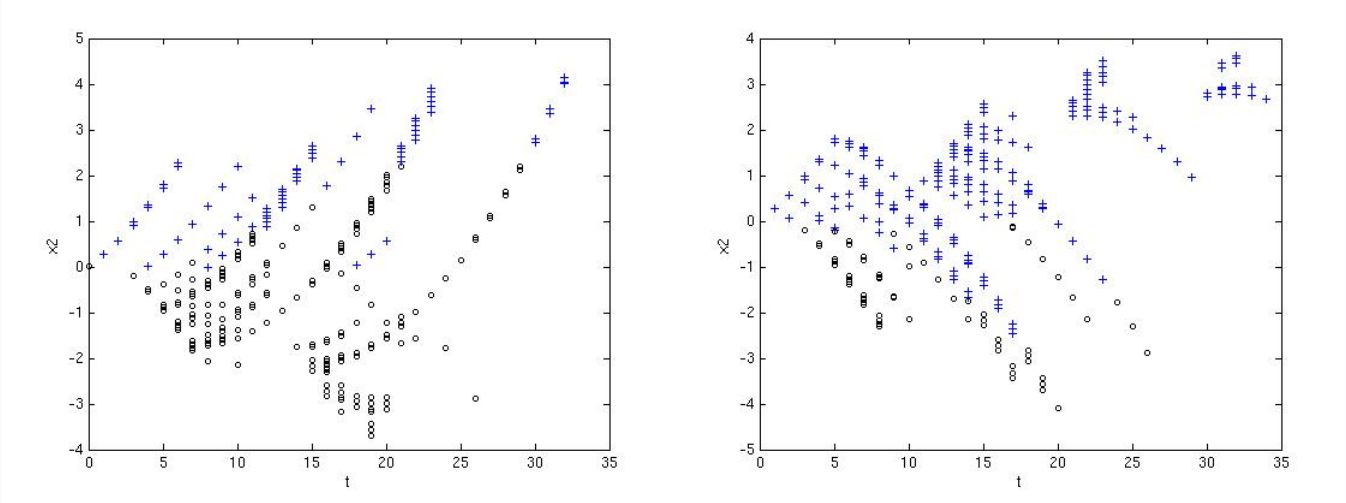




Delta-Sigma circuit (2)

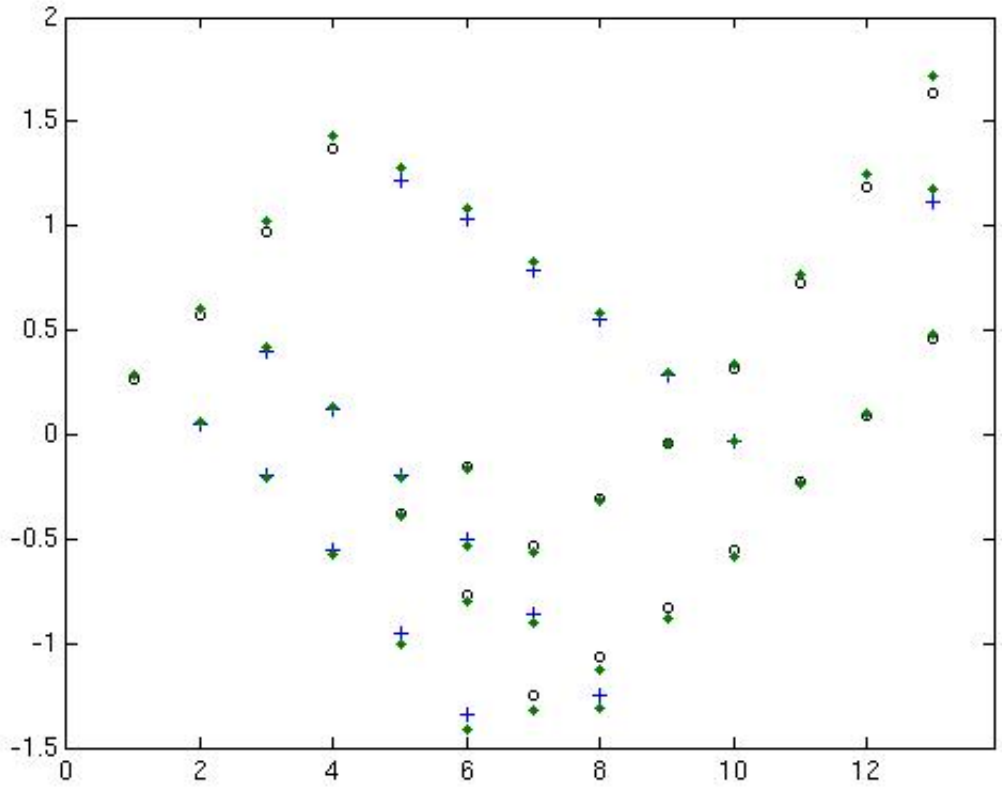
The above hybrid automaton used as the specification automaton. System under test is an implementation of the Delta-Sigma in SPICE netlists.

The observation function $h(x) = 0.1x_1 + 0.2x_2 + 0.5x_3$. The initial state is in $[-0.01, 0.01]^3$ and the input values $u \in [-0.5, 0.5]$.



Delta-Sigma circuit - hybrid state estimation (3)

For a bounded time the implementation is conform to the specification automaton (with $\varepsilon = 1e-2$, no violation of the conformance between the implementation and the specification automaton was detected)



Conclusions

Future Work

- Trace Coverage???

End

Thank You For Your Attention