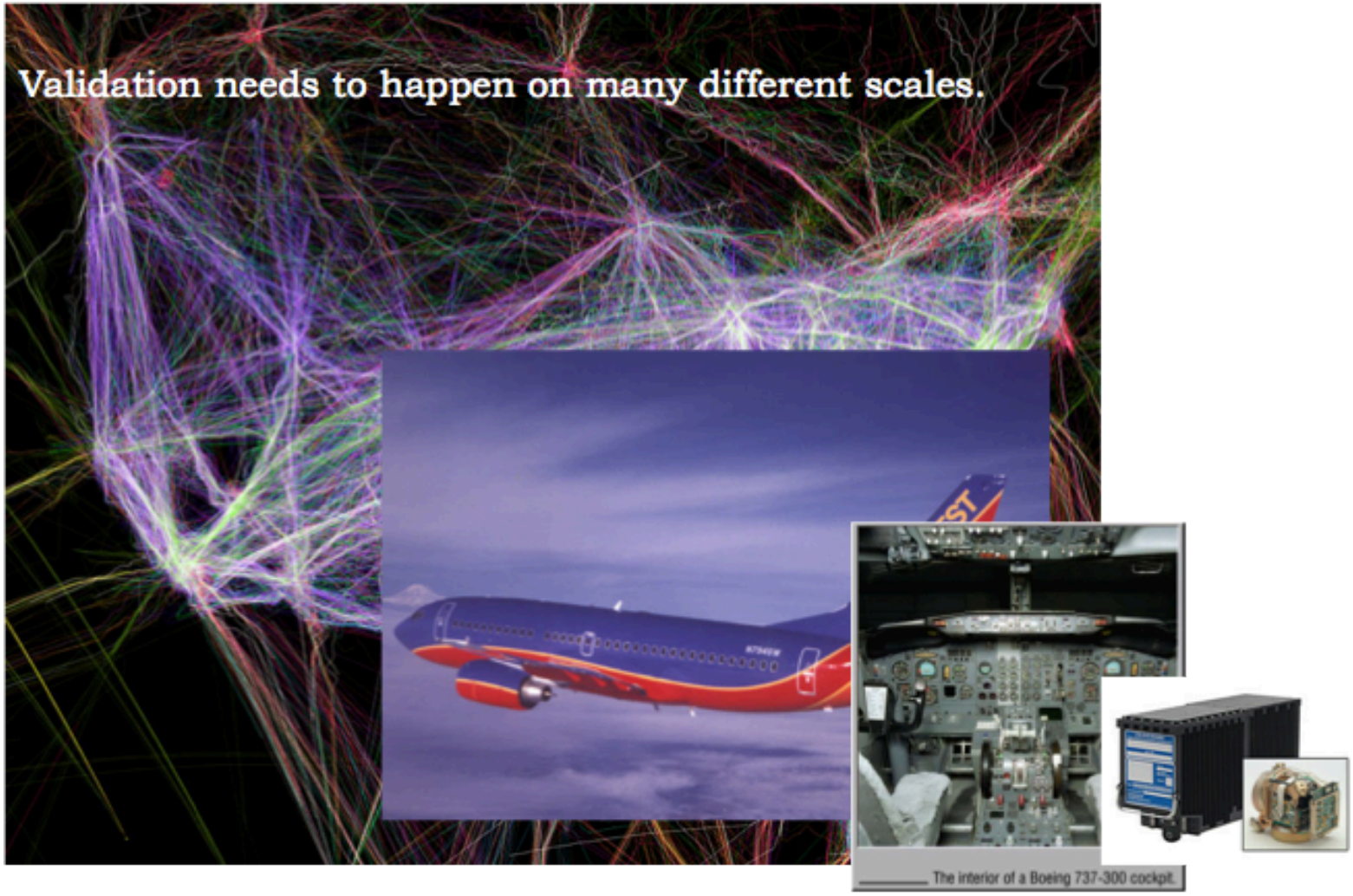
National Aeronautics and Space Administration

Model-Based Validation Testing

Dr. Misty D. Davies
Research Computer Engineer
NASA Ames Research Center

2011 Annual Technical Meeting
May 10–12, 2011
St. Louis, MO

www.nasa.gov



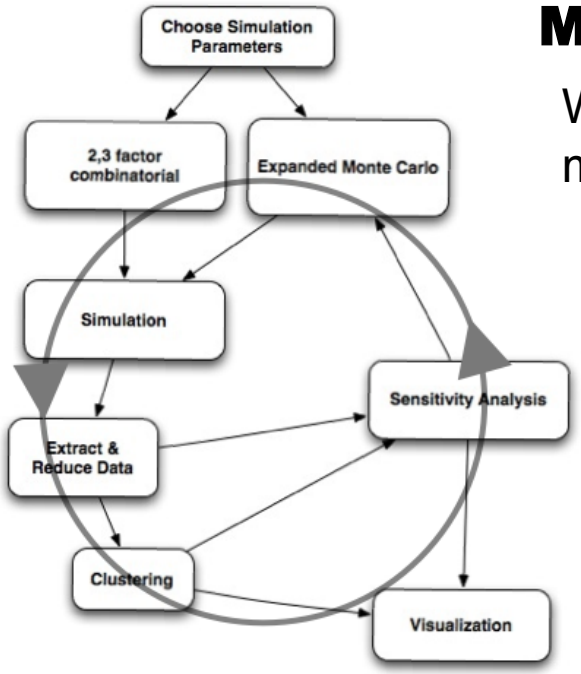
Photos from: http://contrailscience.com/britain-from-above-air-traffic/, www.southwest.com, northrupgrumman.com


6. REQUIREMENTS FOR APPROACHES WITH A MISSED APPROACH LESS THAN RNP 1.0.
a. No single-point-of-failure can cause the loss of guidance compliant with the RNP value associated with a missed approach procedure.
From: FAA Advisory Circular 90-101: Approval Guidance for RNP Procedures with SAAAR. 2005.
QANTAS
Photo from NewZealandView.com

Problem Domain:
Large (thousands of independent variables), complex, non-linear, with interacting modal, continuous, periodic and stochastic parameters.

The Unreasonable Effectiveness of Data
--Alon Halevy, Peter Norvig, and Fernando Pereira

Scene Completion Using Millions of Photographs
--James Hayes, Alexei Efros (CMU)

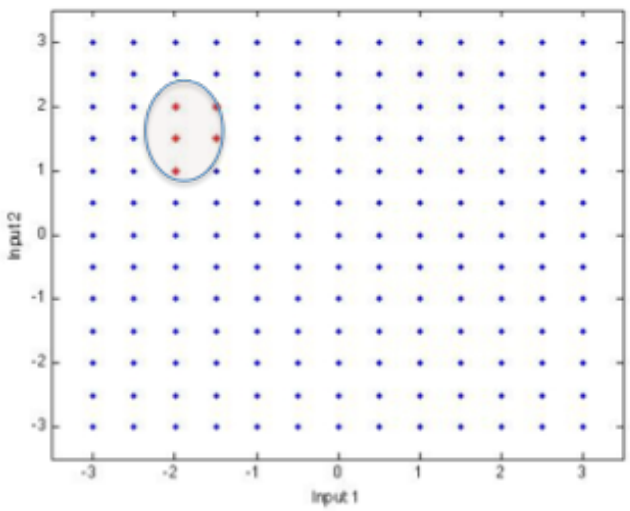


Monte Carlo Filtering

What are the inputs (and their ranges) most likely to lead to some output?

What are we doing differently?

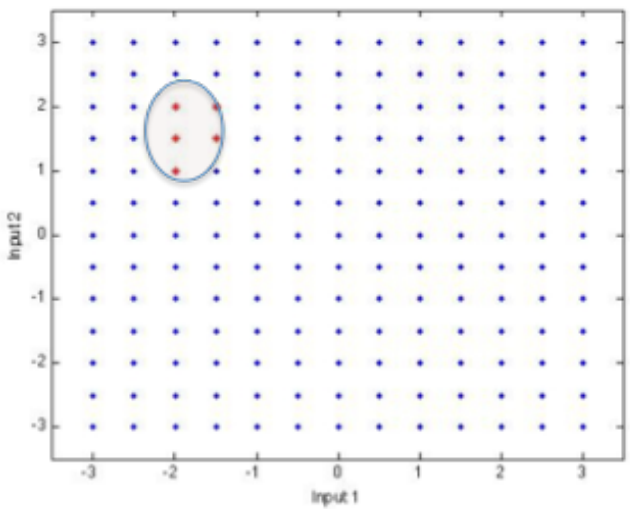
- N-factor combinatorial test case generation
- Data manipulation to identify distributions (continuous, discrete, periodic), reduce dimensionality where it helps, and actually increase dimensionality where we can afford it in order to eliminate the independence assumption.
- Unsupervised learning techniques to find structure.
- Supervised learning techniques (treatment learning) to find dependencies.
- Automated iterations.



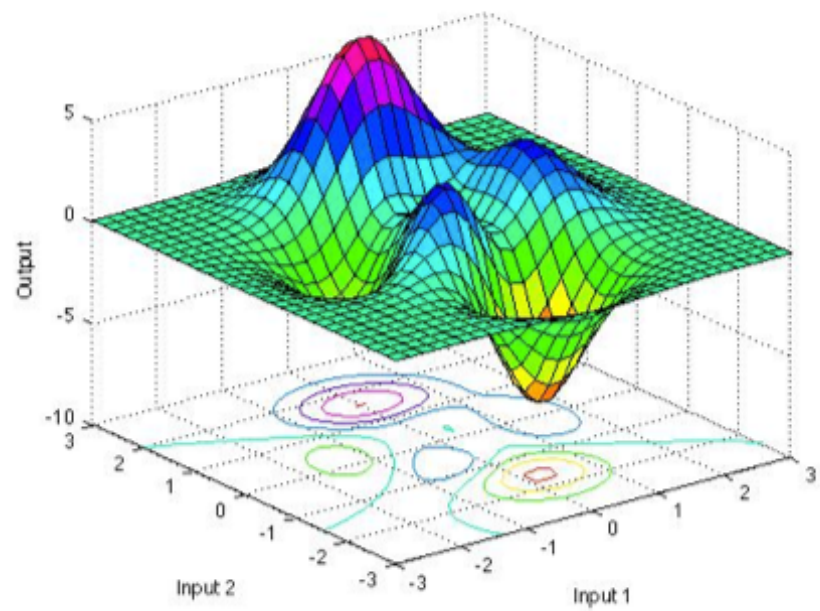
Standard testing casts a grid across the nominal input space, looking for failures.

Most of the space in traditional testing remains uncovered—you know very little about:

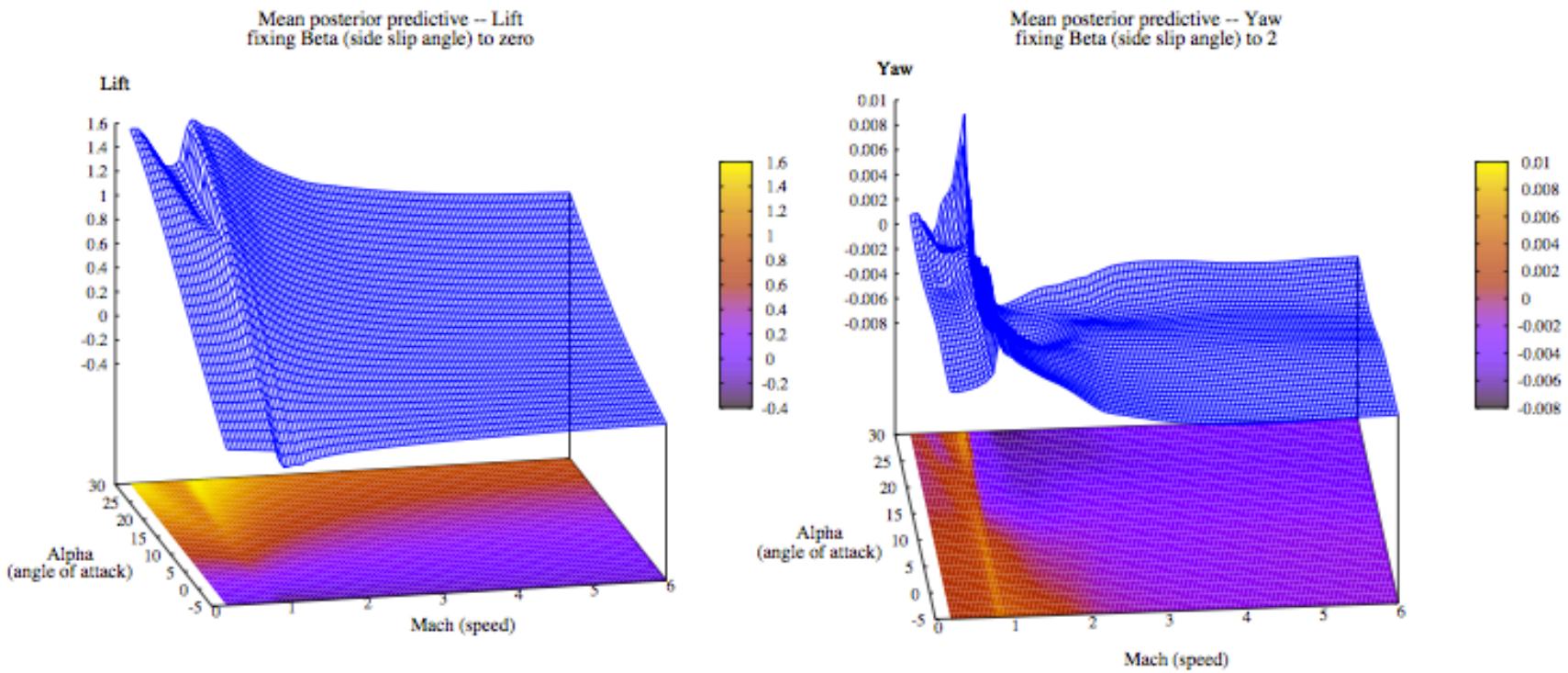
- how likely failures are between data points,
- what the space might look like beyond the nominal input spaces,
- if there is a correlation between your failures and the values of a particular set of variables.



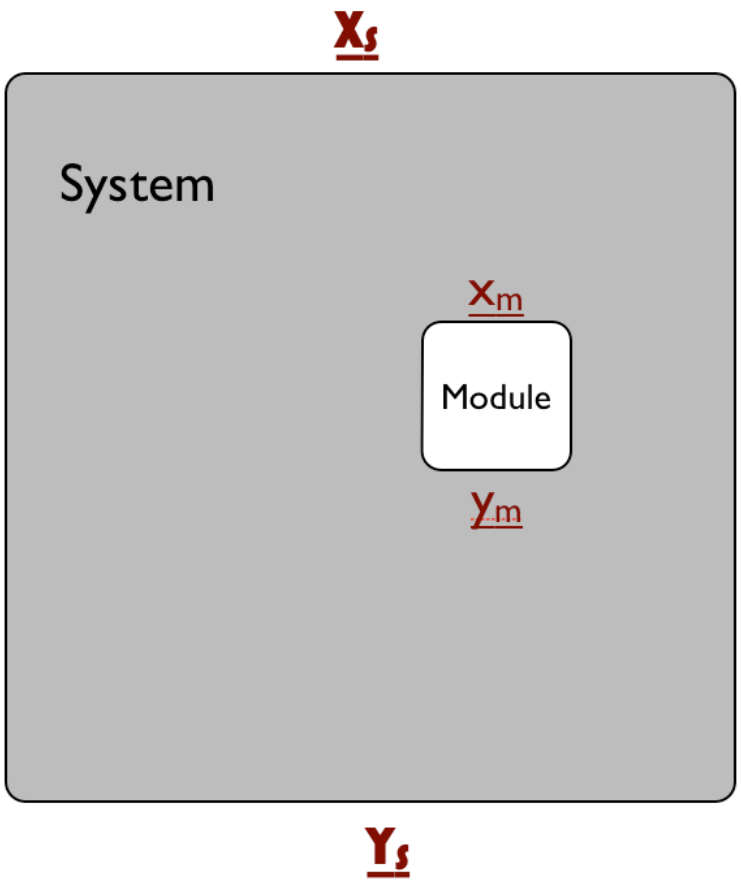
Instead, we build models between the input and output spaces. This means that we can make predictions about where to test in order to find failures, and it can give us a probability for whether or not there are failures in our untested space. It also lets us find patterns in high-dimensional, complex spaces.



How confident can we be in the models we are building?



Surfaces predicting the values of yaw and lift that were built by Bayesian statistical emulation for the Langley Glide-Back Booster. Note that the discontinuity in the surfaces as the booster crosses into the supersonic regime has been captured by the model. (Images courtesy Dr. Herbert Lee, UC Santa Cruz)



Idea: Combine heuristic testing and formal techniques

- Explicit techniques give (some) guarantees
- Heuristic techniques provide scalability

$\underline{y_m}=f(\underline{x_m})$

can be found explicitly using formal techniques

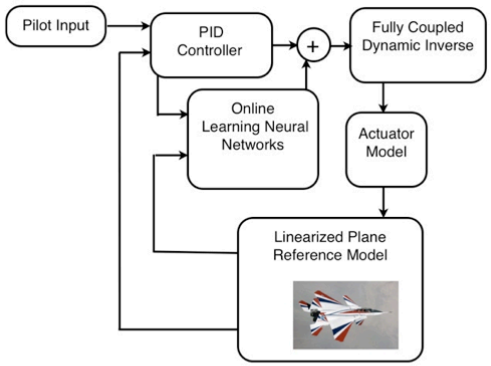
$\underline{Y_s}=f(\underline{X_s})$

$\underline{x_m}=f(\underline{X_s})$

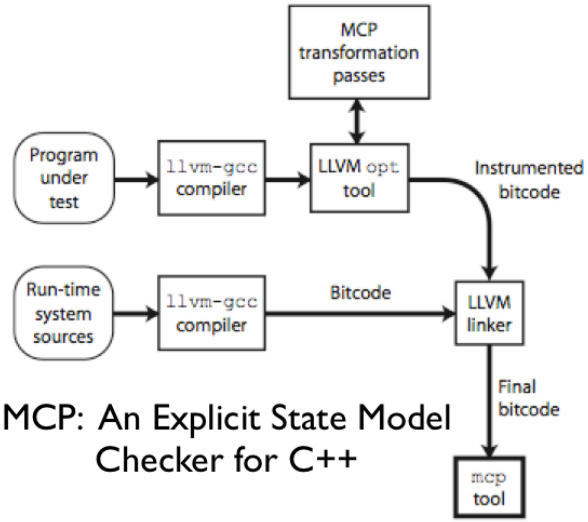
possibly $\underline{Y_s}=f(\underline{y_m})$

can be approximated using heuristic techniques

Testing Plus Model Checking



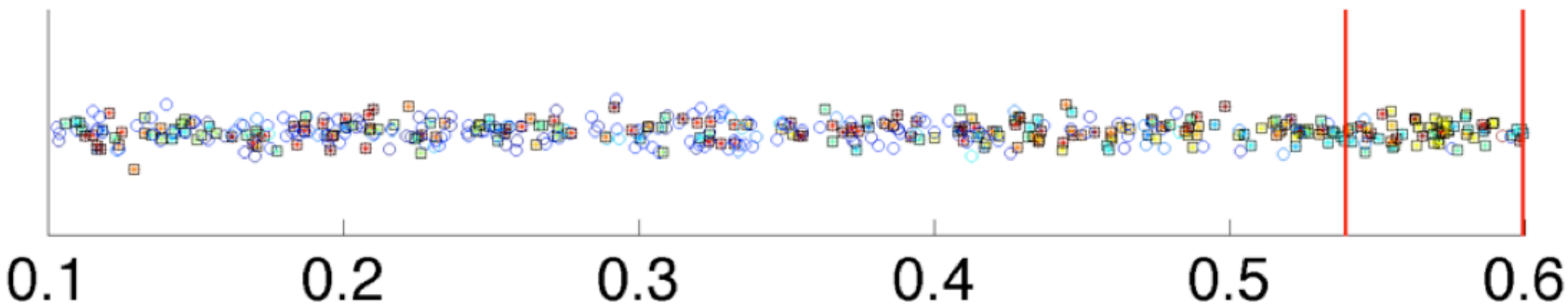
10 seconds
.005 sec/timestep



MCP: An Explicit State Model Checker for C++

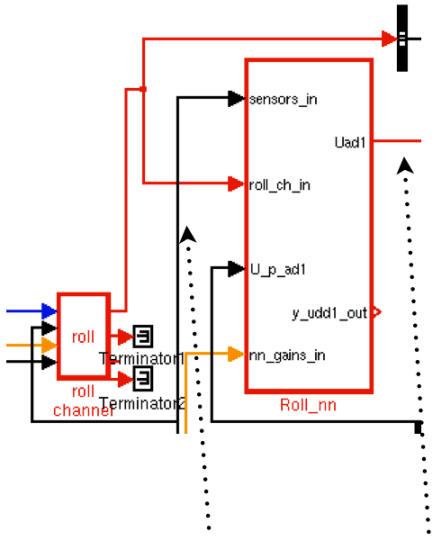
MCP ran out of memory the first time through the OLNNs

With the OLNNs removed (PID control only), MCP ran out of memory after 7 times through the loop.



NAN errors are most associated with roll gains (but the correlation isn't perfect).

Testing Plus Model Checking



Instrumentation

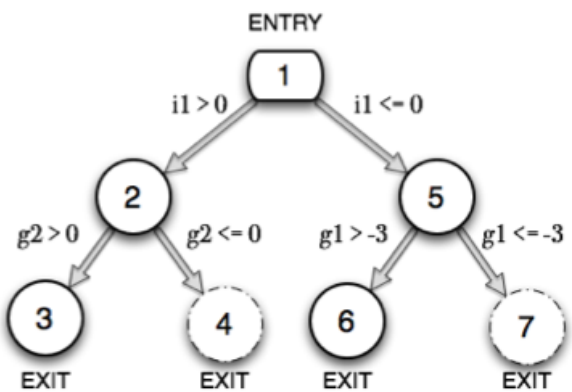
```
---
** starting the model **
--- Step #0
--- rtu_U_p_ad1 = 5.204576e+05 (size 8)
--- mf1 = -7.478910e+01
--- mf1 = -5.205324e+05
--- mf1 = 5.205324e+05
--- mf1 = inf
--- mf1 = inf
--- mf1 = 0.000000e+00
--- mf1 = 0.000000e+00, lb->mf1 = 0.000000e+00
--- u1dd = nan
### GEN2_w_test2_Roll_nn.c(348): Assertion failure: 0
```

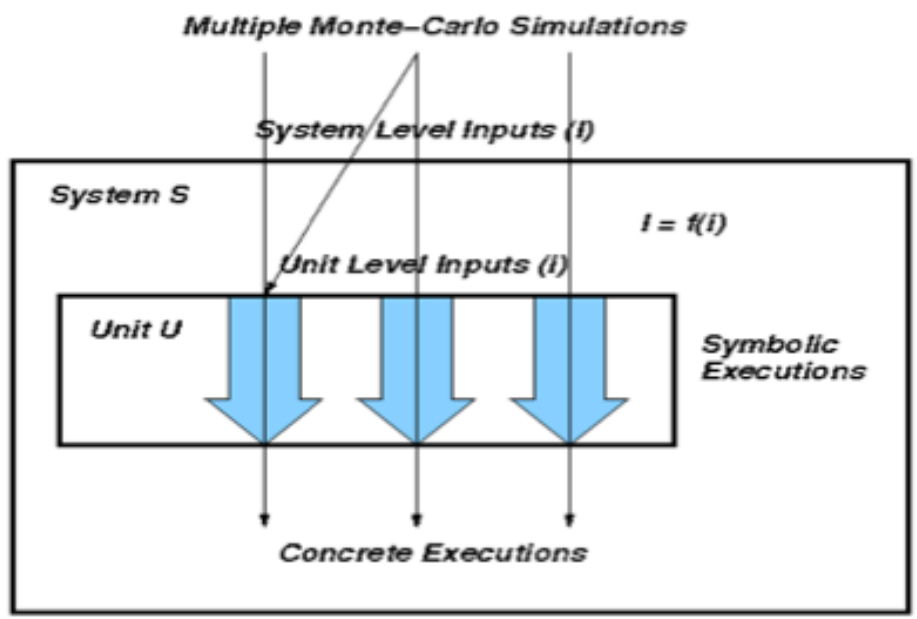
Strategy: Flag the first NAN, treat the inputs to the module for that time step as the test case for the model checker.

```
/* Product: '<S57>/Product4' incorporates:
 *  Constant: '<S54>/cp10'
 *  Constant: '<S55>/cp20'
 */
localB->Product4_a[0] = 1.0000000000000002E-02;
localB->Product4_a[1] = localB->cp11 * 0.1;
localB->Product4_a[2] = localB->cp12 * 0.1;
localB->Product4_a[3] = localB->cp13 * 0.1;

/* Sum: '<S55>/Sum2' */
localB->MathFunction1 = localB->MathFunction1 - rtu_U_p_ad1;
mf1 = mf1 - rtu_U_p_ad1;

/* Gain: '<S55>/Gain' */
localB->MathFunction1 = -localB->MathFunction1;
mf1 = -mf1;

/* Math: '<S55>/Math Function' */
/* Operator : exponential */
localB->MathFunction1 = exp(localB->MathFunction1);
mf1 = exp(mf1);

/* Sum: '<S55>/Sum' incorporates:
 *  Constant: '<S55>/Constant'
 */
localB->Sum = 1.0 - localB->MathFunction1;
```

Testing Plus Concolic Execution

```
int g1 = 1, g2 = 2;
int System(int I1, int I2)
{
  if (I1 > 0) g1 = I2; else g1 = -I2;
  g2 = I1 + 3;
  Unit(I2, I1);
}
int Unit(int i1, int i2)
{
  if(i1 > 0) {
    i2 = g2;
    if(i2 > 0) return 0; else return 1;
  } else {
    i2 = g1 + 3;
    if(i2 > 0) return 2; else return 3;
  }
}
```





Testing Plus Concolic Execution

System Inputs (I): Pt, Ps, Alt

Subsonic:

$$Ma = \sqrt{5\left[\left(\frac{P_t}{P_s}\right)^{\frac{0.4}{1.4}} - 1\right]}$$

Supersonic:

$$\frac{P_t}{P_s} = \left(\frac{5.76Ma^2}{5.6Ma^2 - 0.8}\right)^{3.5} \frac{2.8Ma^2 - 0.4}{2.4}$$

Unit Inputs (i): Ma, Cf, Cfbterm, Cfterm

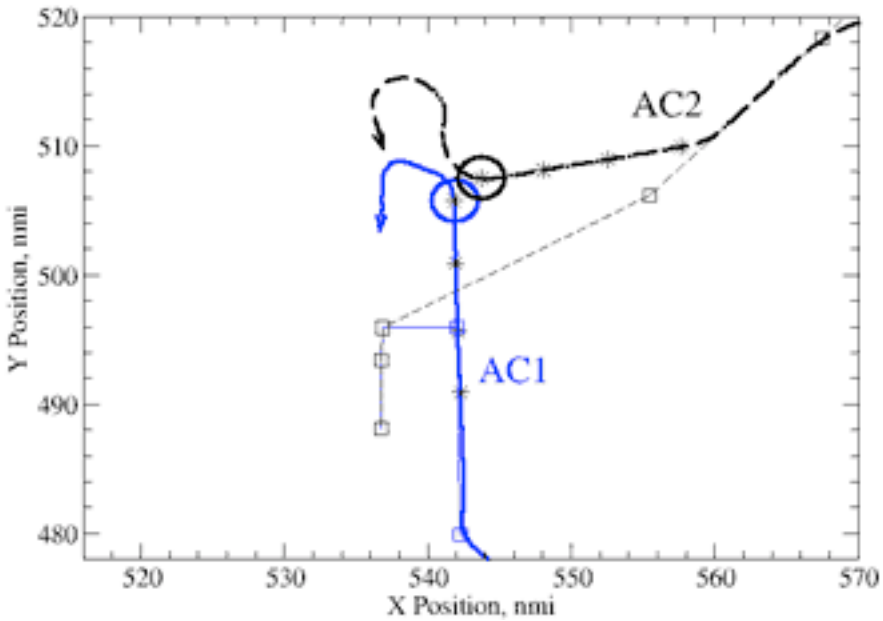
```
[Tree]
(Cf > CfTerm) (C, ...)
  (Ma >= (780000 / 1000000)) (C, ...)
    (Ma > (1040000 / 1000000)) (C, ...)
      (Ma >= (600000 / 1000000)) (C, ...)
        (Cfb > CfbTerm) (C, ...)
          (Ma >= 1) (C, ...)
            (Ma <= (2000000 / 1000000)) (C, ...)
            (Ma > (2000000 / 1000000)) (C, ...)
          (Ma < 1) (S, ...)
        (Cfb <= CfbTerm) (S, ...)
      (Ma < (600000 / 1000000)) (S, ...)
    (Ma <= (1040000 / 1000000)) (S, ...)
  (Ma < (780000 / 1000000)) (S, ...)
(Cf <= CfTerm) (S, ...)
```

After 25 tests—

N-factor:
16 covered,
10 uncovered

Model-based:
21 covered,
12 not covered

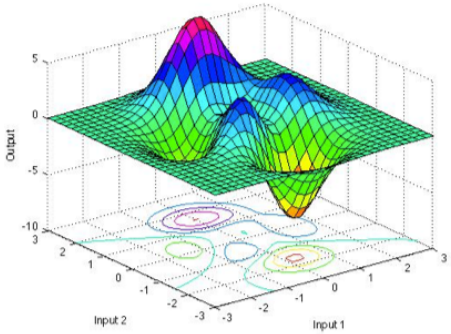
Terminal TSAFE



This figure represents a sample operational error. It shows several minutes of track data for two aircraft (AC1 and AC2) leading up to the loss of separation (LOS) where the two circles, which are 3 nautical miles in diameter, overlap. The asterisks are minute markers denoting three minutes before the LOS. The sharp turn to the right of AC2 after LOS is due to controller intervention.

When should you use model-based testing?

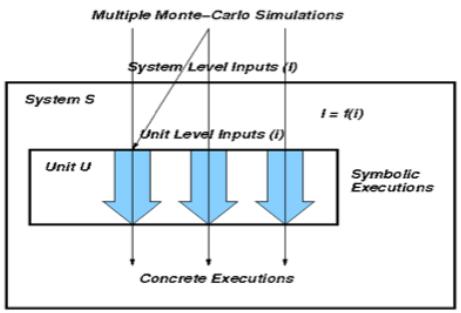
When you don't have lots of information about relationships (too complex, too many) but you do have lots of data (and the ability to get more data).



When you have some idea of what 'good' data or 'bad' data look like.

Even better if:

You can isolate some parts of the system to 'white-box' and can feed the the illuminated data in as information.



Model-Based Validation Testing

Why change current testing methods?

Finding bugs in modern flight software is an arduous process. Interdependencies between the systems and subsystems of the flight vehicle and the environment mean that, not only does each component need to be explored, the interfaces between the components must also be exercised. During the later parts of the design phase the system is usually validated using high-fidelity simulations. Advances in computing power make it possible to try many thousands of combinations of input variables to exercise the behavior of the system. However, a thorough exploration of the entire flight envelope results in gigabytes of data that requires expert domain knowledge to interpret; and because of the high dimensionality of the data, it is difficult to pick out patterns with the human eye.

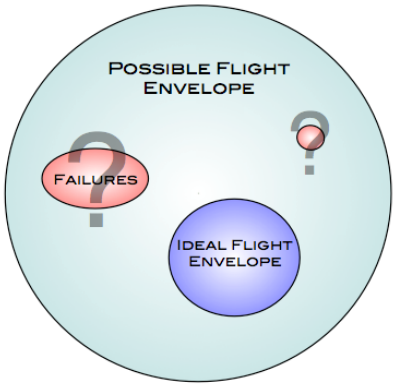
Additionally, standard testing is based on isolated operating points for the system. Inputs are selected randomly, or are chosen according to some grid, and are usually concentrated in the ideal flight envelope. The outputs of the system are evaluated against failure criteria, with each test case either 'succeeding' or 'failing' - there is no notion of a distance to failure.

How is model-based validation testing better than traditional testing methods?

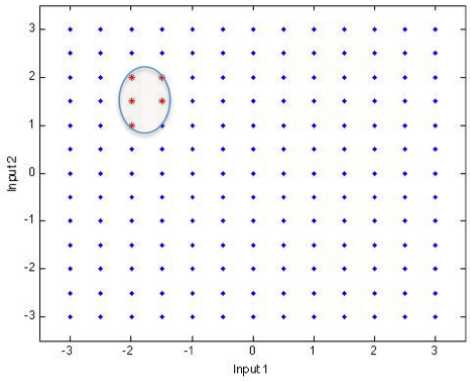
Instead of trying to find isolated failures, model-based validation testing uses collected test data to build a relationship between the inputs and the outputs. We can then treat testing as a scientific experiment. The model we are building becomes a hypothesis for the system behavior, and we can test and refine our hypothesis using standard methods from experimental design. When the model we are building represents the actual system closely enough, we can use it to make predictions about the system in places we haven't tested yet. More importantly, we can begin to put bounds on the maximum error between reality and our model using statistics and formal methods. These means that we can produce evidence that provides a high confidence of system safety.

What do you mean when you say that you treat testing like an experiment?

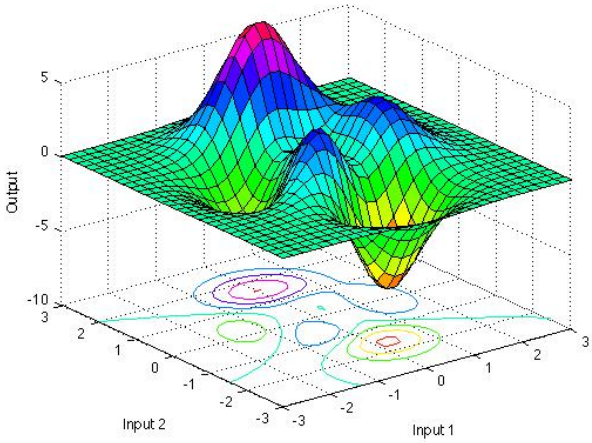
The end goal of our work is to be able to make *predictions* about the behavior of complex systems. In order to do that, we build a model of the system. We build the model



The possible flight envelope is much larger than the ideal flight envelope. Standard testing concentrates on the ideal flight envelope. Model-based testing allows us to explore off-nominal but still possible scenarios in order to find failures.



Standard testing looks at isolated test points and only evaluates whether a case is a success or a failure.



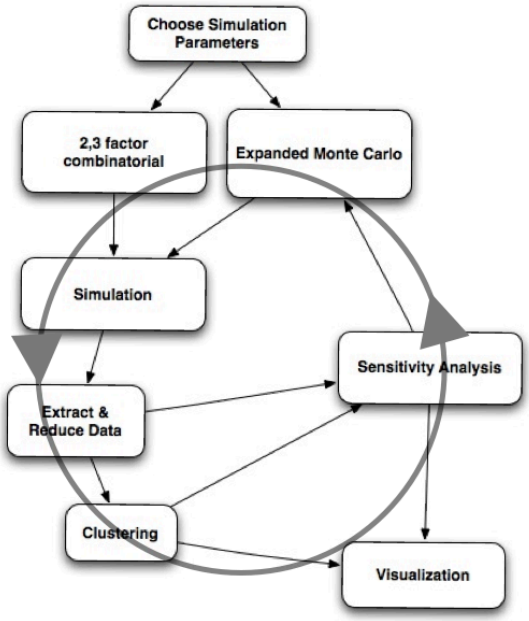
Model-based testing allows us to predict regions of success or failure, and enables us to bound our uncertainty.

Model-Based Validation Testing

automatically by making *observations* of the system, and using machine learning techniques to determine the relationships between the things we can control and the system behavior. For our work, we use unsupervised machine learning techniques to find structure in the system. This tells us what form our model should take, and it can automatically find anomalies in the data. In order to build an accurate model quickly, we borrow ideas from experimental design like *factorial experiments* - most failures are only triggered by one, two, or three factors at most, and we build test suites that take advantage of that fact.

We use each successive model as a *hypothesis*, and *test* the predictions we can make. These tests allow us to refine the models until we have reached some minimum confidence bound.

Whenever possible, we also can incorporate information gleaned from *formal methods* techniques. These are mathematical proofs that some part of the system must behave in a particular way. We also can use model-based validation testing to give good initial models to formal methods tools, enabling the tools to more quickly produce definitive evidence of safety.



The model-based testing process automates the creation and refinement of hypotheses.

When do you use model-based validation testing, instead of other techniques?

Model-based validation testing is designed for use in large and/or complex systems with a combination of continuous and discrete variables. You should be able to run tests on the system, and, ideally you should be able to define whether a particular behavior was 'good' or 'bad'. Most of the time, the notion of 'good' and 'bad' behavior comes from the system goals. Model-based validation testing has previously been used for launch pad abort scenarios, small satellite design, and medical device design. We are currently using model-based validation testing to explore next generation (NextGen) air traffic management concepts of operation.

Recent papers:

G. Gay, T. Menzies, M. Davies, and K. Gundy-Burlet. "Automatically finding the control variables for complex system behavior." Automated Software Engineering 17(4), December 2010, pp 439-468.

M. Davies, G. Limes and K. Gundy-Burlet. "A Hardware Model Validation Tool for Use in Complex Systems." AIAA Space Conference, Anaheim, CA, Aug. 31-Sep. 2, 2010

S. Thompson, M. Davies and K. Gundy-Burlet. "Hybrid Decompositional Verification for Discovering Failures in Adaptive Flight Control Systems." AIAA Infotech@Aerospace Conference, Atlanta, GA, Apr. 20, 2010

K. Gundy-Burlet, J. Schumann, T. Barrett, and T. Menzies, "Parametric Analysis of a Hover Test Vehicle Using Advanced Test Generation and Data Analysis," *AIAA Aerospace*, 2009.

J. Schumann, K. Gundy-Burlet, C. Pasareanu, T. Menzies, and T. Barrett. "Tool Support for Parametric Analysis of Large Software Systems", *Proceedings of Automated Software Engineering, 23rd IEEE/ACM International Conference,* 2008.

For more information, please contact:

Misty Davies
(650) 604-0476 misty.d.davies@nasa.gov