mystic: rigorous model certification and engineering design under uncertainty

a framework for highly-constrained non-convex optimization and UQ

mystic

a framework for highly-constrained
non-convex optimization
and uncertainty quantification

Mike McKerns

http://www.uqfoundation.org

motivation: automated design of materials

- what is the molecular structure that best produces the desired materials properties?
  - can we optimize the probability that a structure will produce the desired properties within some tolerance?
- what is the optimal reaction path between two molecular configurations?
  - can we optimize the probability that a transition will occur between the initial and final states?
- can we address these questions directly?
  - can we formulate a quantity of interest (QOI) as a "goodness" metric, where we can use any and all information to constrain an automated search that considers only the space of viable solutions?

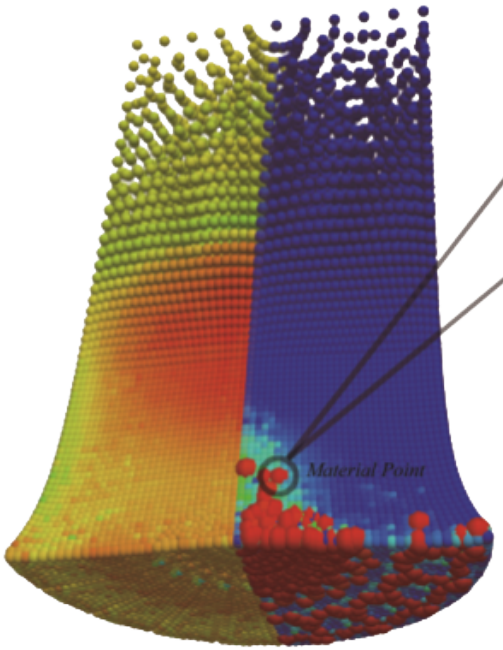
CALTECH
PSAAP

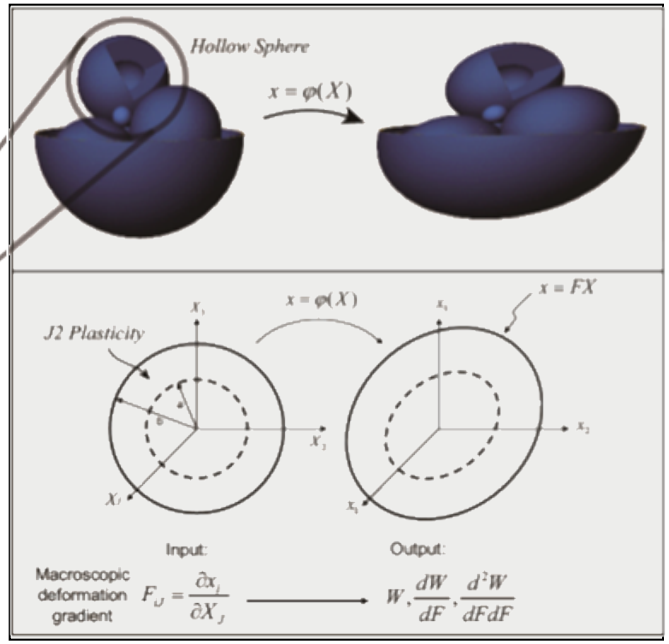
ExMatEx
Extreme Materials at Extreme Scale

typical problem: model of impact plasticity

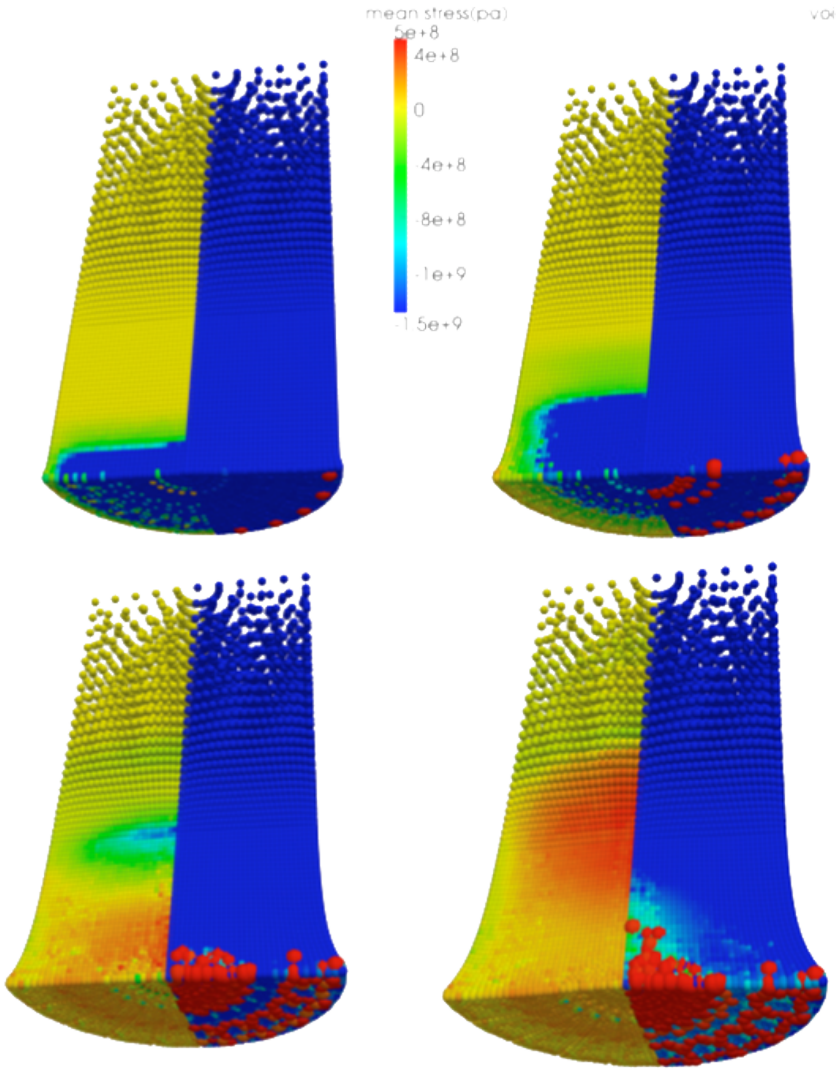
- A 1100-0 aluminum cylinder is impacted axially against a rigid wall at velocity of 300m/s. [D.E. Grady and M.E. Kipp, 1993]
- The cylinder is discretized by material points, each material point represents a hollow sphere discretized by finite elements (FE). J2 Isotropic plasticity model with rate-dependent power law hardening and thermal softening, equation of state in Mie-Gruneisen type is used for the subscale hollow sphere model.

Axial section of the end part of an aluminum alloy bar impacted on a rigid wall



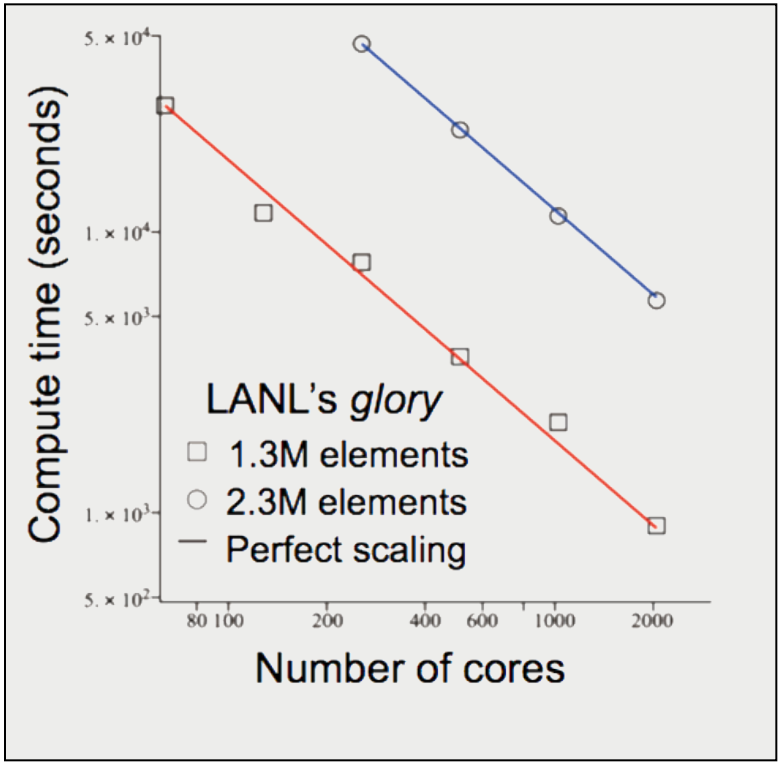


a realistic model can be very expensive



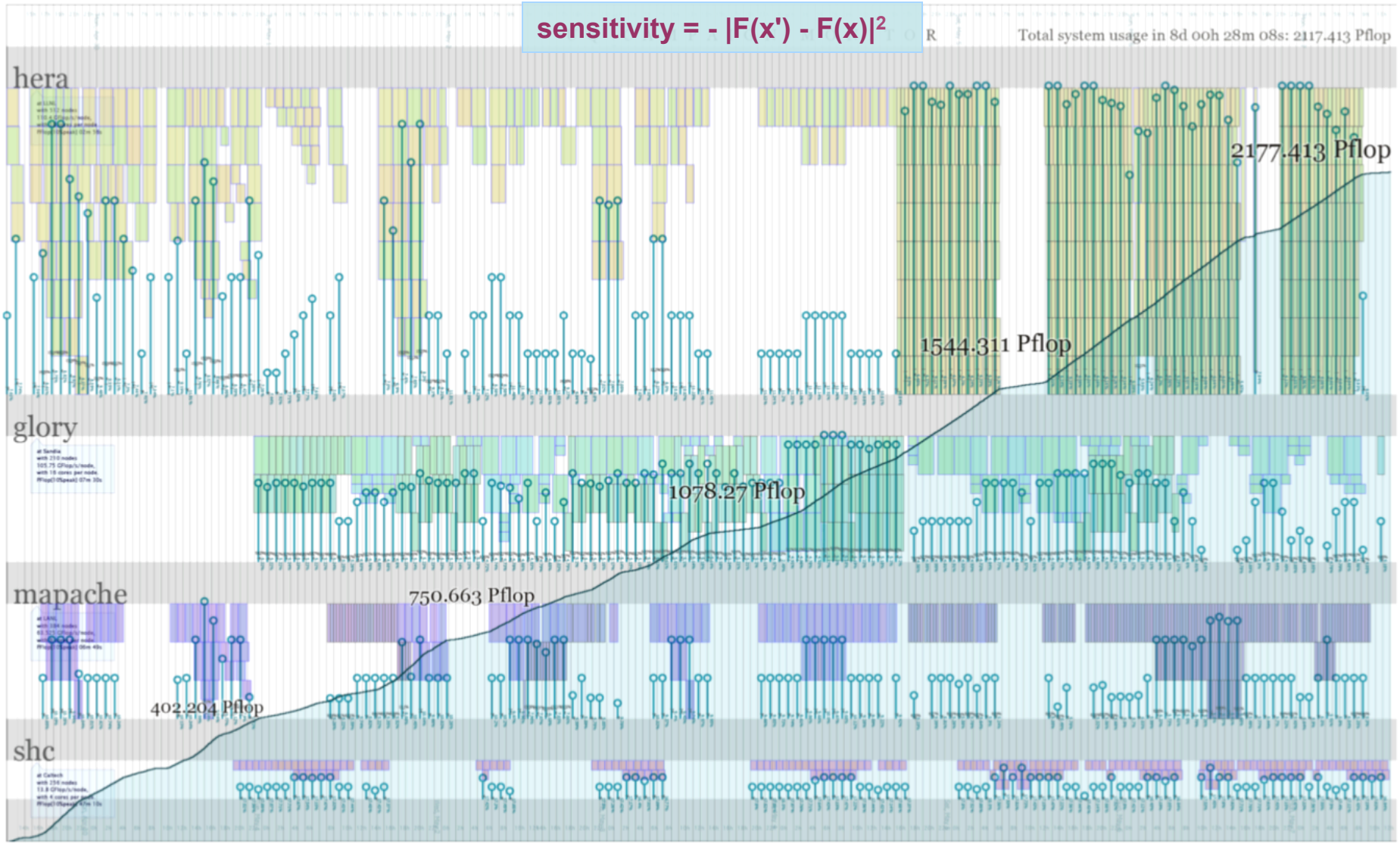
Taylor test, porous Aluminum (FE^2)

- Hybrid MPI/Pthreads
- Multiscale material models: near-perfect MPI scalability
- Pthreads implementation: Theoretical scalability limit



optimization can be very very expensive

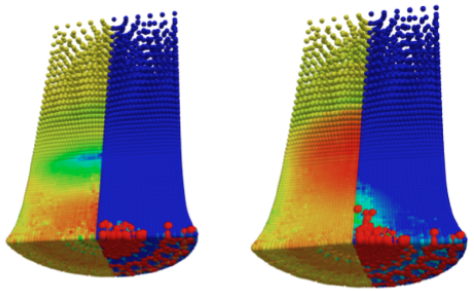
CALTECH PSAAP



UQ run over 8 days over distributed resources

the need for optimizer-based parallelism

- cluster-scale objective functions require:
  - large-scale parallel and distributed computing
  - asynchronous execution
  - monitoring, caching, archiving, and job restarts
  - dynamic decision making
  - efficient batch execution of the model



Taylor test, porous Aluminum (FE^2)

- optimizers, however, tend to be:
  - notoriously serial (iterative process)
  - (synchronously) bound to a model
  - diagnostic-limited and stateless
  - of fixed execution strategy
  - efficient in solver algorithmic speed

…the opposite of what is required!

```
# the function to be minimized
# and initial values
import taylor as my_model
x0 = [0.8, 1.2, 0.7]

# obtain the solution
import diffev as solver
solution = solver(my_model, x0)
```

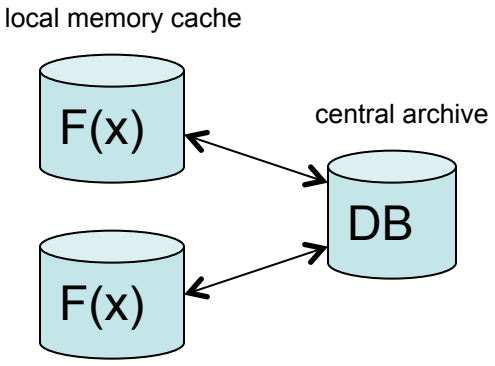
optimization tends to be blocking until a solution is found

parallel graph execution and statefulness

PATHOS
a framework for parallel graph management and execution in heterogeneous computing

```
# the function to be minimized and the bounds
from mystic.models import rosen as my_model
lb = [0.0, 0.0, 0.0]; ub = [2.0, 2.0, 2.0]

# get monitor and termination condition objects
from mystic.monitors import LoggingMonitor
stepmon = LoggingMonitor(1, 'log.txt')
from mystic.termination import ChangeOverGeneration
COG = ChangeOverGeneration()

# select the parallel launch configuration
from pyina.launchers import TorqueMpi
my_map = TorqueMpi('25:ppn=8').map

# instantiate and configure the nested solver
from mystic.solvers import PowellDirectionalSolver
my_solver = PowellDirectionalSolver(len(lb))
my_solver.SetStrictRanges(lb, ub)
my_solver.SetEvaluationLimits(1000)

# instantiate and configure the outer solver
from mystic.solvers import BuckshotSolver
solver = BuckshotSolver(len(lb), 200)
solver.SetRandomInitialPoints(lb, ub)
solver.SetGenerationMonitor(stepmon)
solver.SetNestedSolver(my_solver)
solver.SetSolverMap(my_map)
solver.Solve(my_model, COG)
# obtain the solution
solution = solver.bestSolution
```

- available launchers:
  - multiprocess, threaded
  - MPI parallel
  - RPC/IPC (distributed)
  - SSH
  - GPU, cloud
- available schedulers:
  - torque, slurm, lsf
- hierarchical maps can be built with a coupling strategy

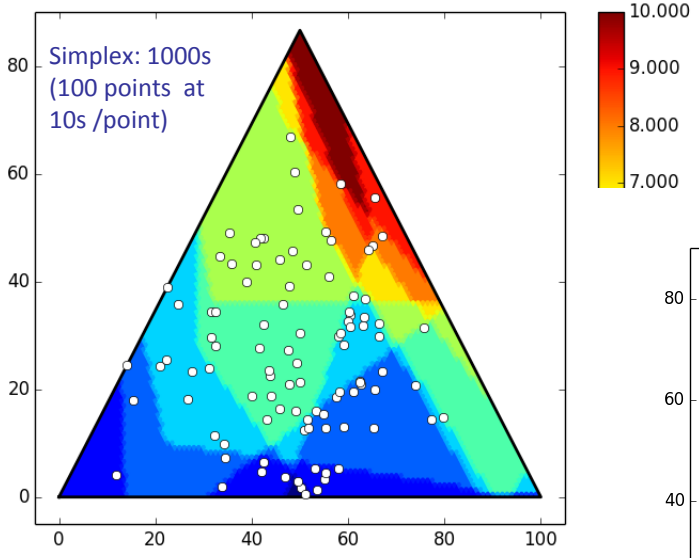


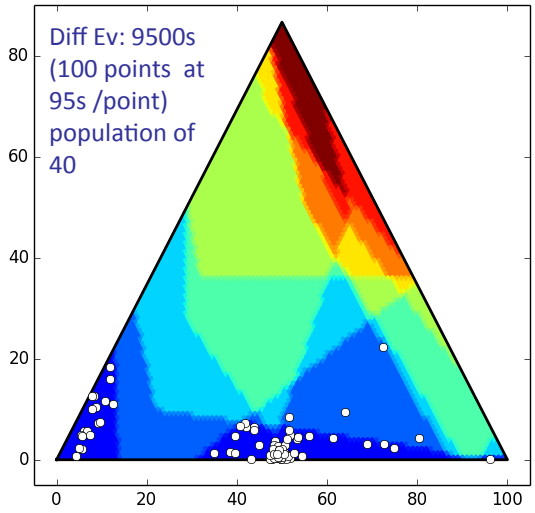
automated state saving and sharing

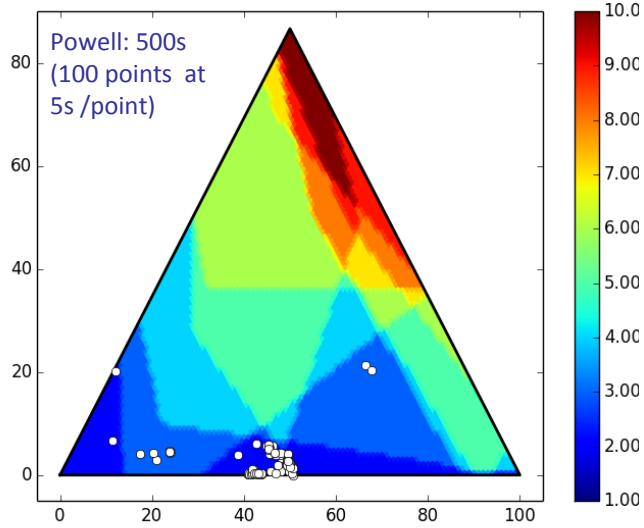
cache-to-archive interaction

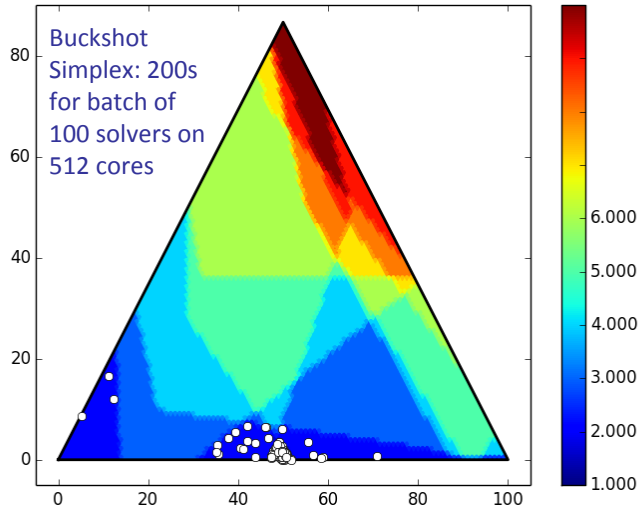
caching to memory, hdf, file, directory, database

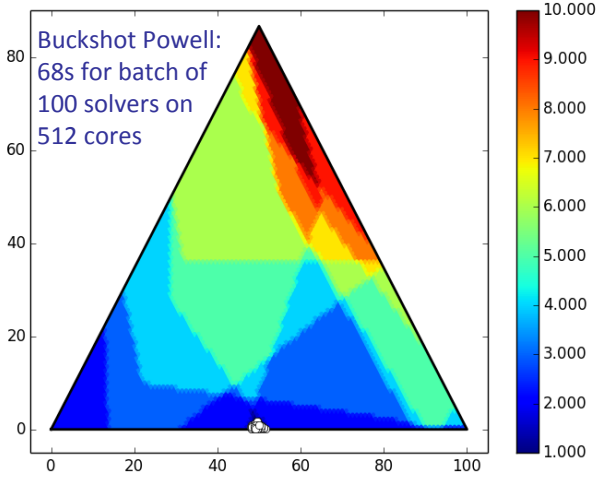
massively-parallel ensemble optimizers





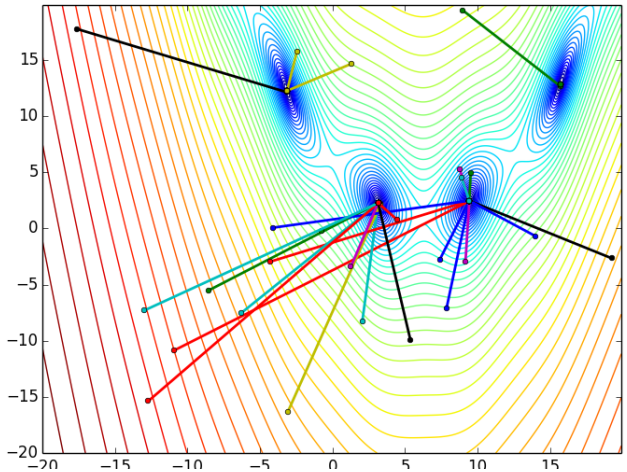




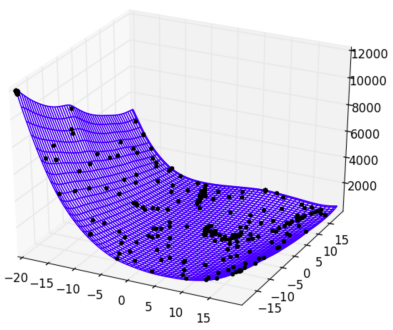


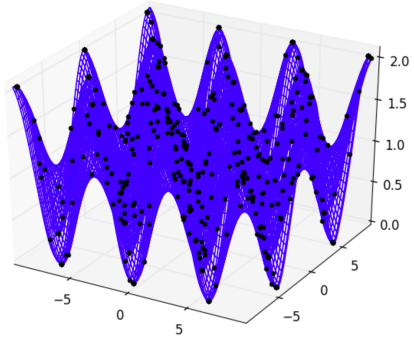
BROOKHAVEN
NATIONAL LABORATORY

ensemble global search and interpolation

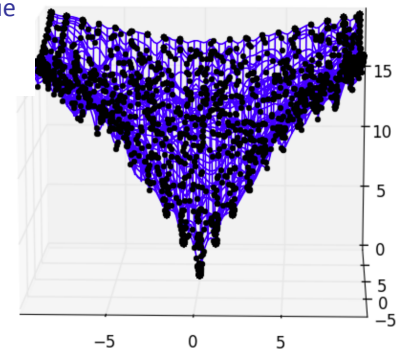


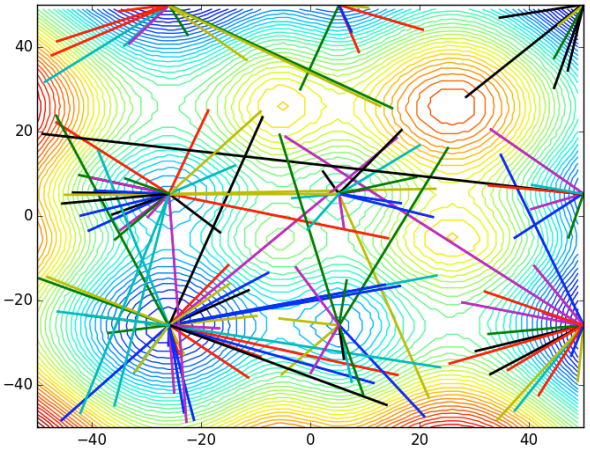
Single Buckshot Powell search for all minima





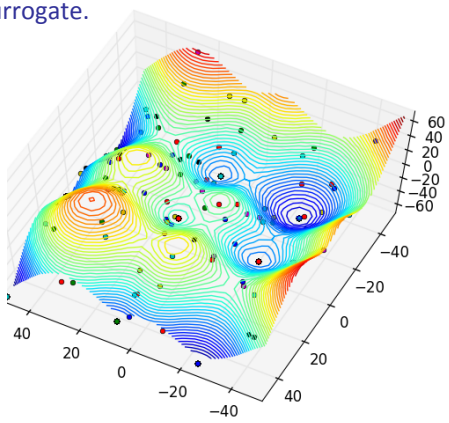
interpolated surfaces due to search for extrema and/or critical points





Two-iteration Buckshot Powell search for all minima.

Interpolate points to build a surrogate.



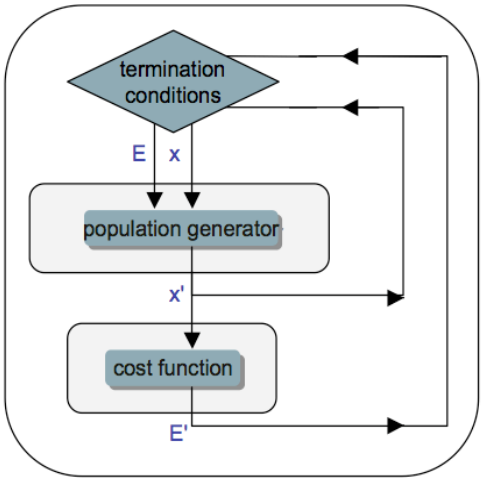
```
dude@hilbert>$ python global_search.py
CacheInfo(hit=17, miss=8, load=0, maxsize=None, size=8)
CacheInfo(hit=24, miss=1, load=0, maxsize=None, size=9)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=9)
CacheInfo(hit=25, miss=0, load=0, maxsize=None, size=9)
min: -70.8861291838 (count=1)
pts: 9 (values=8, size=9)
```

"cache" in this case is an abstraction on storage. "load" is local memory cache, while "hit" is an archive hit. "miss" is a new point. Results shown are for when configured for direct connectivity with archival database.

scalable: dynamic asynchronous execution

```
# the function to be minimized and initial values
import taylor as my_model
x0 = [0.8, 1.2, 0.7]

# get monitor and termination condition objects
from mystic.monitors import VerboseMonitor
stepmon = VerboseMonitor(5)
from mystic.termination import ChangeOverGeneration
terminate = ChangeOverGeneration()

# instantiate and configure the solver
from mystic.solvers import DifferentialEvolutionSolver
solver = DifferentialEvolutionSolver(len(x0))
solver.SetSaveFrequency(100, 'solver.pkl')
solver.SetInitialPoints(x0)
solver.SetGenerationMonitor(stepmon)
solver.SetObjective(my_model)
solver.SetTermination(terminate)
solver.Solve()

# obtain the solution
solution = solver.bestSolution
```

- plug-and-play components:
  - monitoring, logging, caching
  - population generators, parallelism
  - penalties, constraints, stop conditions



an optimizer is composed of a population generator and termination conditions, acting on a cost function

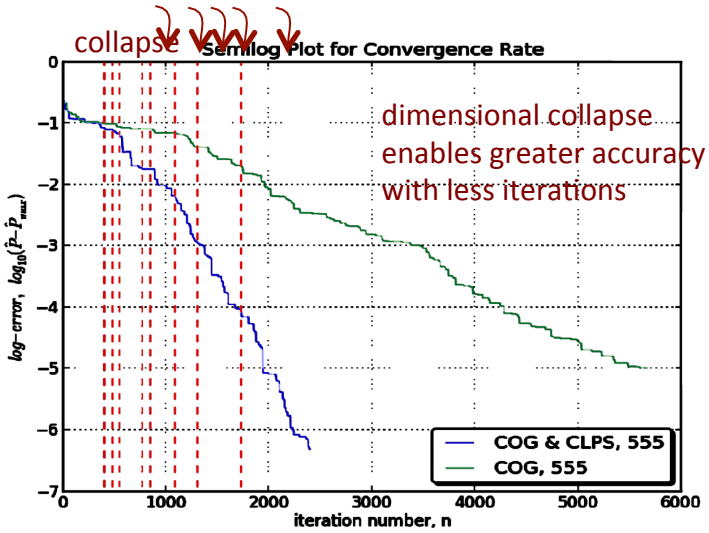
a cost function provides a difference metric
- $E = |F(x) - G|^2$

an optimizer has state

```
# obtain diagnostic information
function_evals = solver.evaluations
iterations = solver.generations
cost = solver.bestEnergy

# modify the solver configuration; restart
from mystic.termination import VTR, Or
COG = ChangeOverGeneration(tolerance=1e-8)
solver.SetTermination(Or(VTR(),COG))
solver.Step()
solver.Step()

# obtain the current best solution
solution = solver.bestSolution
```

transforms: simplify and reduce search space

- box (range) constraints
- nonlinear (functional) constraints
- uniqueness and set-membership constraints
- probabilistic and statistical constraints
- constraints imposing sampling statistics
- inputs from sampling distributions
- constraints from legacy data (points and data sets)
- constraints from models and distance metrics
- constraints on (product) measures
- support vector (weight, independence) collapse




```
>>> from mystic.constraints import unique, discrete, integers, with_mean, and_, not_
>>> from mystic.math.measures import mean
>>> c = and_(unique, discrete(range(10,100,3))(lambda x: x), with_mean(50)(lambda x:x))
>>> c([6,33,14,33,51])
[89.0, 44.0, 50.0, 32.0, 35.0]
>>> mean(_)
50.0
>>> c = and_(integers()(lambda x:x), not_(lambda x:[0]*len(x)), with_mean(0)(lambda x:x))
>>> c([6,3,-1,-3,5])
[4, 1, -3, -5, 3]
>>> mean(_)
0.0
```

constraints may be solved by nested optimizations

penalties and constrained optimization

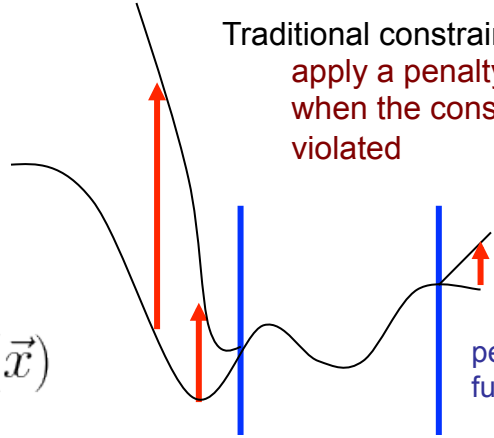
mystic
a framework for highly-constrained non-convex optimization and uncertainty quantification

```
from mystic.math.measures import mean, spread
from mystic.constraints import with_penalty, with_mean
from mystic.constraints import quadratic_equality

# build a penalty function
@with_penalty(quadratic_equality, kwds={'target':5.0})
def penalty(x, target):
  return mean(x) - target

# define an objective
def cost(x):
  return abs(sum(x) - 5.0)

# solve using a penalty
from mystic.solvers import fmin
x = array([1,2,3,4,5])
y = fmin(cost, x, penalty=penalty)
```

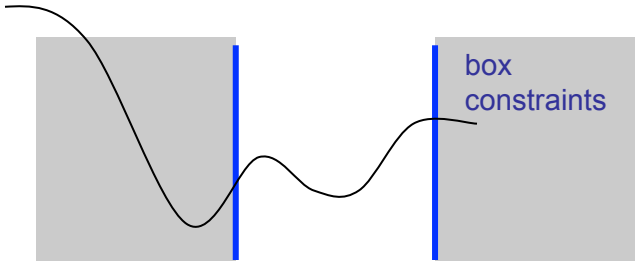
$$\phi(\vec{x}) = f(\vec{x}) + k \cdot p(\vec{x})$$

fast, but implicit, inaccurate, and can add spurious features

Traditional constraints methods apply a penalty to the cost when the constraints are violated



Decoupling constraints often produces a convex optimization for the QOI



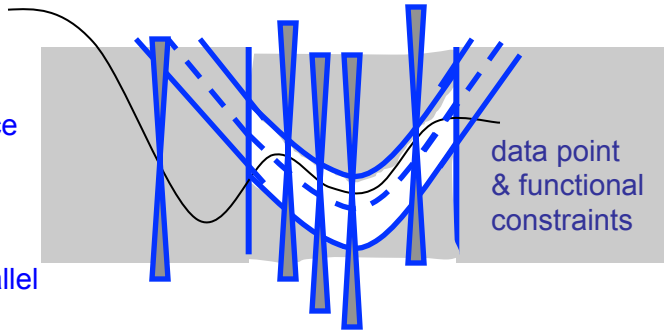
```
# build a kernel transformation
@with_mean(5.0)
def constraint(x):
  return x

# solve using constraints
y = fmin(cost, x, constraints=constraint)
```

$$\phi(\vec{x}) = f(c(\vec{x}))$$

explicit and can be parallelized, can strongly reduce search space

$$|\Psi' > = \hat{c}|\Psi >$$

operators that commute can be spawned in parallel



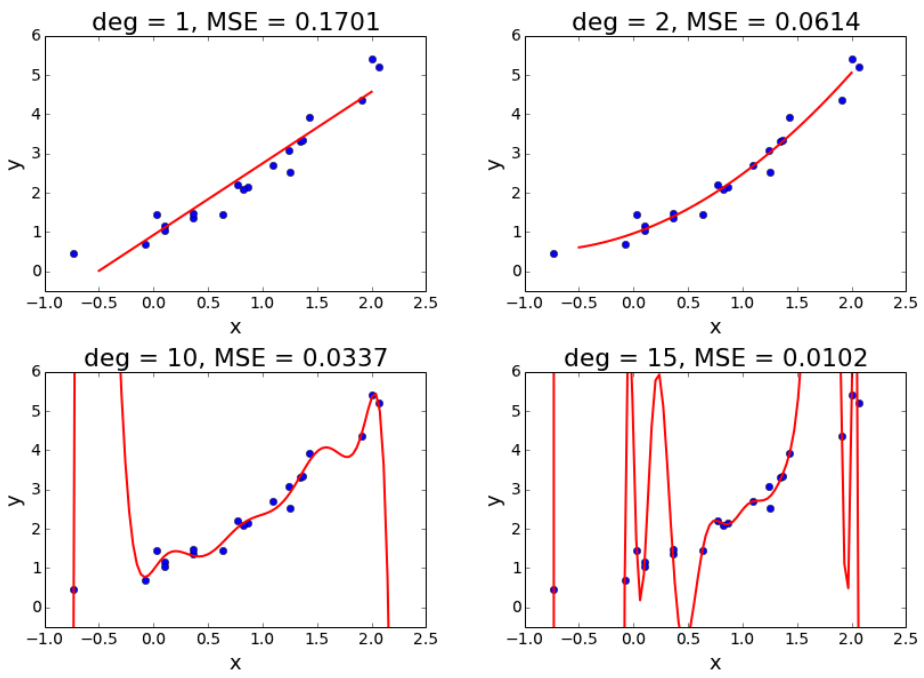
example: global MIP with symbolic constraints

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 88
         Function evaluations: 3560
[ 6.  0.  8.  4.  9.  3.  9.]
```

```
def objective(x):
    return 0.0

bounds = [(0,10)]*7
# constraints
equations = """
98527*x0 + 34588*x1 + 5872*x2 + 59422*x4 + 65159*x6 - 1547604 - 30704*x3 - 29649*x5 == 0.0
98957*x1 + 83634*x2 + 69966*x3 + 62038*x4 + 37164*x5 + 85413*x6 - 1823553 - 93989*x0 == 0.0
900032 + 10949*x0 + 77761*x1 + 67052*x4 - 80197*x2 - 61944*x3 - 92964*x5 - 44550*x6 == 0.0
73947*x0 + 84391*x2 + 81310*x4 - 1164380 - 96253*x1 - 44247*x3 - 70582*x5 - 33054*x6 == 0.0
13057*x2 + 42253*x3 + 77527*x4 + 96552*x6 - 1185471 - 60152*x0 - 21103*x1 - 97932*x5 == 0.0
1394152 + 66920*x0 + 55679*x3 - 64234*x1 - 65337*x2 - 45581*x4 - 67707*x5 - 98038*x6 == 0.0
68550*x0 + 27886*x1 + 31716*x2 + 73597*x3 + 38835*x6 - 279091 - 88963*x4 - 76391*x5 == 0.0
76132*x1 + 71860*x2 + 22770*x3 + 68211*x4 + 78587*x5 - 480923 - 48224*x0 - 82817*x6 == 0.0
519878 + 94198*x1 + 87234*x2 + 37498*x3 - 71583*x0 - 25728*x4 - 25495*x5 - 70023*x6 == 0.0
361921 + 78693*x0 + 38592*x4 + 38478*x5 - 94129*x1 - 43188*x2 - 82528*x3 - 69025*x6 == 0.0
"""

from mystic.symbolic import generate_penalty, generate_conditions
pf = generate_penalty(generate_conditions(equations))

from numpy import round as nround


if __name__ == '__main__':

    from mystic.solvers import diffev2
    result = diffev2(objective, x0=bounds, bounds=bounds, penalty=pf,
                     constraints=nround, npop=40, gtol=50, disp=True, full_output=True)
```

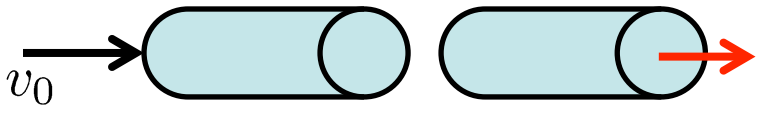
over-fitting can be avoided with better tools

- over-fitting is a consequence of having missing information that fails to inform the optimizer



- can come from:
  - missing or approximate constraining information
  - abstract convergence criteria (scoring function)
  - poor objective function
- no over-fitting if:
  - all constraining information is explicitly applied (by kernel transformation)
  - objective and scoring function are the QOI and our actual quality metric

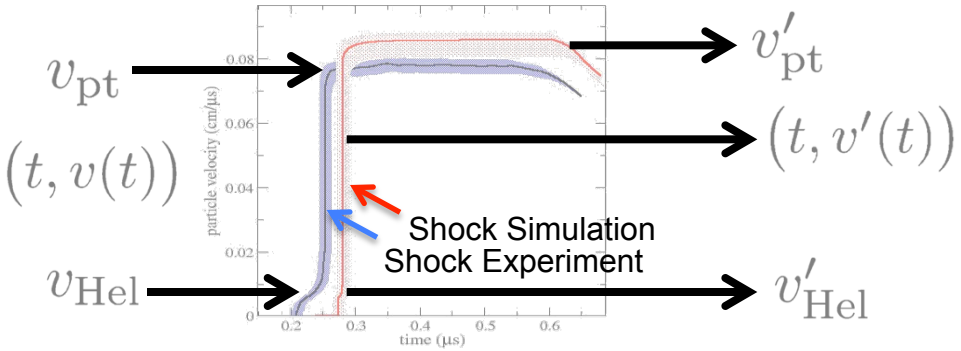
UQ objectives: "how good is my model?"



model error (for the entire curve)

$$d\Big((t \to v(t))(x), (t \to v'(t))(x, \lambda)\Big)$$

we want to estimate **we can compute**



model error (for a scalar QOI)

$$\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x, \lambda)\right|$$

We don't know the exact microstructure and chemical composition, these are stochastic in nature

Geometric features have some degree of randomness

We have incomplete information on the distribution of x

We know v_0, h, r only up to some tolerance

$h \in [h_{\min}, h_{\max}]$, $\mathbb{E}[h] = m$, $\text{Var}(h) \leq \sigma$

We have incomplete information on the distribution of microstructure and chemical composition

Volume fractions of iron, carbon, ...

Average grain orientation and size,correlation between grain orientations as a function of distance, ...

statistical error

$$\mathbb{E}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x, \lambda)\right|^2\right]$$

model uncertainty

$$\mathbb{P}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x, \lambda)\right| \geq a\right]$$

likelihood of failure

$$\mathbb{P}\left[\left|v_{\text{Hel}}(x) - v'_{\text{Hel}}(x, \lambda)\right| \geq a\right] \leq \varepsilon$$

needle in an infinite-dimensional haystack

optimal bound on the statistical error

$$\sup_{\mu\in\mathcal{A}} \mathbb{E}_{\mu}\left[\left|v_{\mathrm{Hel}}(x) - v'_{\mathrm{Hel}}(x,\lambda)\right|^2\right]$$

the optimal model is obtained by minimizing the optimal bound on the statistical error

$$\inf_{v'_{\mathrm{Hel}}} \sup_{\mu\in\mathcal{A}} \mathbb{E}_{\mu}\left[\left|v_{\mathrm{Hel}}(x) - v'_{\mathrm{Hel}}(x,\lambda)\right|^2\right]$$

optimal design requires nested infinite-dimensional optimizations

- min/max nested optimizations of all possible outcomes over all possible models of the material
  - coordinate space provides a huge combinatorial problem
  - with an optimization over all possible outcomes of the above

- in terms of all possible states, the configurations that provide the desired properties are usually quite rare

how to solve it? simplify, simplify, simplify

~~optimal bound on the statistical error~~ $\sup_{\mu\in\mathcal{A}} \mathbb{E}_\mu \left[\left| v_{\mathrm{Hel}}(x) - v'_{\mathrm{Hel}}(x,\lambda) \right|^2 \right]$

~~the optimal bound on~~ the optimal model is obtained by minimizing the statistical error

$$\inf_{v'_{\mathrm{Hel}}} \sup_{\mu\in\mathcal{A}} \mathbb{E}_\mu \left[\left| v_{\mathrm{Hel}}(x) - v'_{\mathrm{Hel}}(x,\lambda) \right|^2 \right]$$

- select the "most likely" distribution
  - collapse optimization to the sub-manifold of coordinate space
- select the form of the model
  - collapse to a single parameterized model or class of models

- solve a difficult finite-dimensional machine learning problem
- then typically use monte-carlo estimation to obtain bounds

OUQ: a robust uncertainty theory

$$\mathcal{A} := \left\{ (g,\mu) \;\middle|\; \begin{array}{c} (g\colon \mathcal{X} \to \mathbb{R}, \mu \in \mathcal{P}(\mathcal{X})) \text{ is consistent with} \\ \text{all given information about the real system } (G,\mathbb{P}) \\ (\textit{e.g.}\ \text{legacy data, first principles, expert judgement}) \end{array} \right\}.$$

- Optimal bounds on the quantity of interest $\mathbb{E}_{X\sim\mathbb{P}}[q(X,G(X))]$ (optimal w.r.t. the information encoded in $\mathcal{A}$) are found by minimizing/maximizing $\mathbb{E}_{X\sim\mu}[q(X,g(X))]$ over all admissible scenarios $(g,\mu)\in\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) \le \mathbb{E}_{X\sim\mathbb{P}}[q(X,G(X))] \le \mathcal{U}(\mathcal{A}),$$

where $\mathcal{L}(\mathcal{A})$ and $\mathcal{U}(\mathcal{A})$ are defined by the minimization and maximization problems

extremes are bound by information in the form of constraints

$$\mathcal{L}(\mathcal{A}) := \inf_{(g,\mu)\in\mathcal{A}} \mathbb{E}_{X\sim\mu}[q(X,g(X))],$$

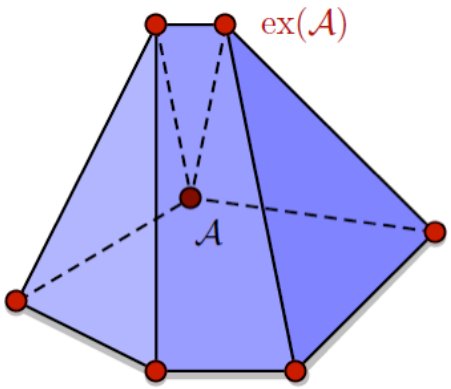
formulated to handle UQ for catastrophic rare-events

$$\mathcal{U}(\mathcal{A}) := \sup_{(g,\mu)\in\mathcal{A}} \mathbb{E}_{X\sim\mu}[q(X,g(X))].$$

OUQ reduces infinite to finite dimensional

optimal bound on the statistical error

$$\sup_{\mu\in\mathcal{A}} \mathbb{E}_{\mu}\left[\left|v_{\mathrm{Hel}}(x) - v'_{\mathrm{Hel}}(x,\lambda)\right|^2\right]$$



OUQ problems reduce to searches over finite dimensional families of extremal scenarios of $\mathcal{A}$

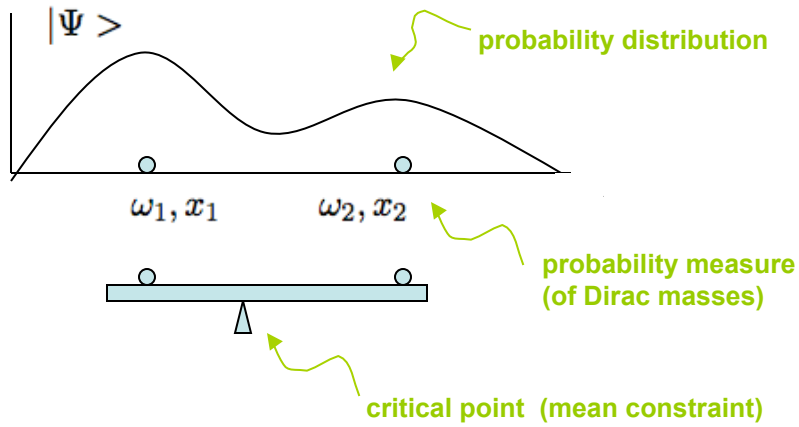
The dimension of the reduced problem is proportional to the number of probabilistic inequalities that describe $\mathcal{A}$

- optimization is over product measures
  - does not require selection of a probability distribution
  - imposes constraining information on the possible distributions
- valid only when solving for extrema
  - this is what we want when solving for worst-case bounds
  - much cheaper and more rigorous than monte carlo sampling

UQ with unknown probability distributions

- min/max on probability measure space (not input parameter space)

$$|\Psi' > = \hat{c}|\Psi > = \sum_i \omega_i |x_i >$$

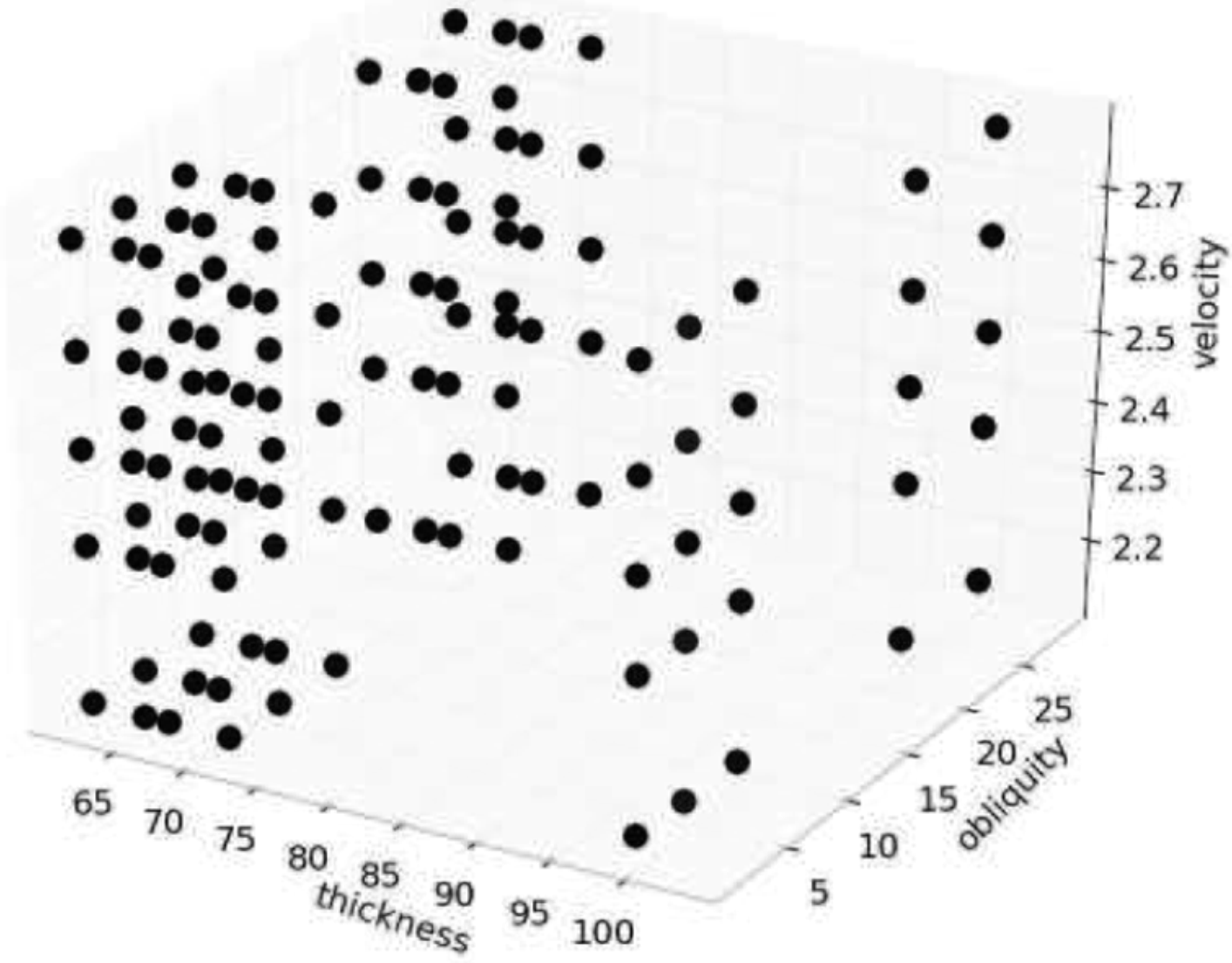


- mean-constrained optimization balances weights and positions of Dirac masses around a critical point

how many points are required? N+1 or less, where N is the number or constraints.

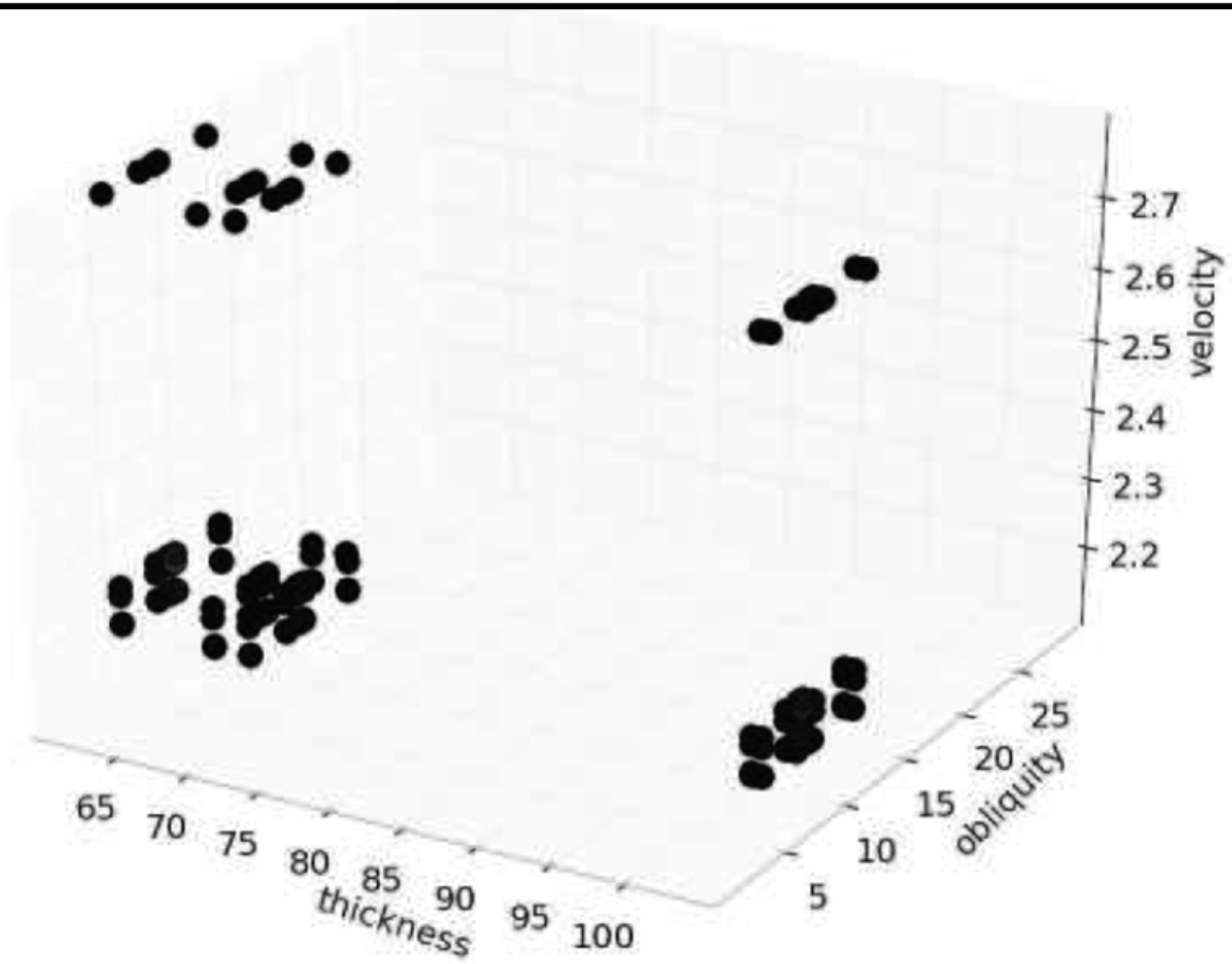
- OUQ is an optimization problem to find the rigorous bounds on system behavior
  - all information is captured as constraints
  - constraints restrict the set of all possible solutions (by directly constraining solution space)
  - systems with minimal to no experimental data or unobserved rare events that govern system behavior
- instead of selecting a "best" model or distribution or prior, we can optimize over all possible models, distributions, or priors.
  - selecting a model or distribution is treated as an assumption or information (i.e. a constraint)
  - our "prior" step becomes one of quantifying all the knowledge we have about the problem, and then encoding that knowledge as constraints

initial representation of probability distribution…



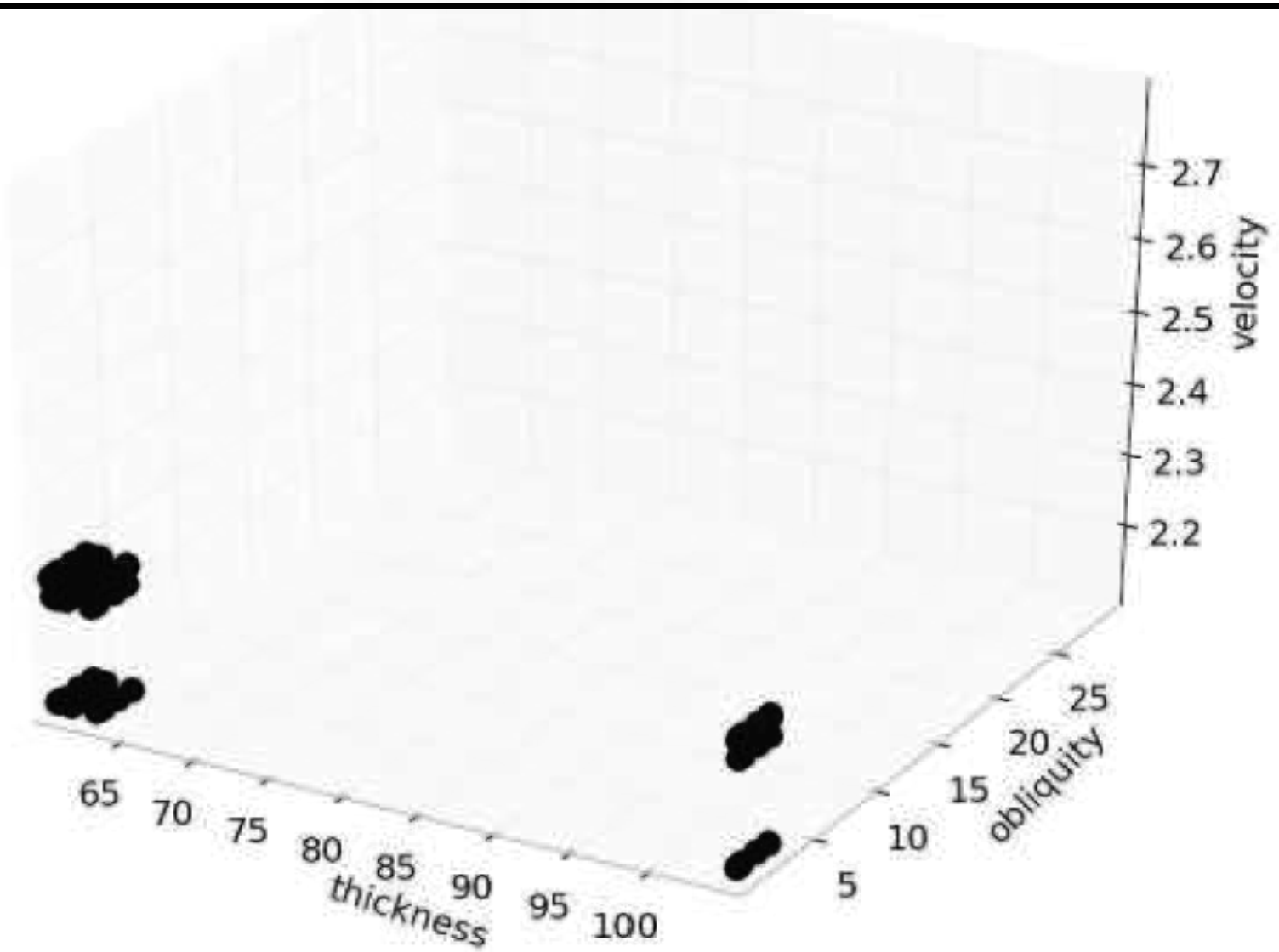
Support Points at iteration 0

…solver looks for extremal cases…



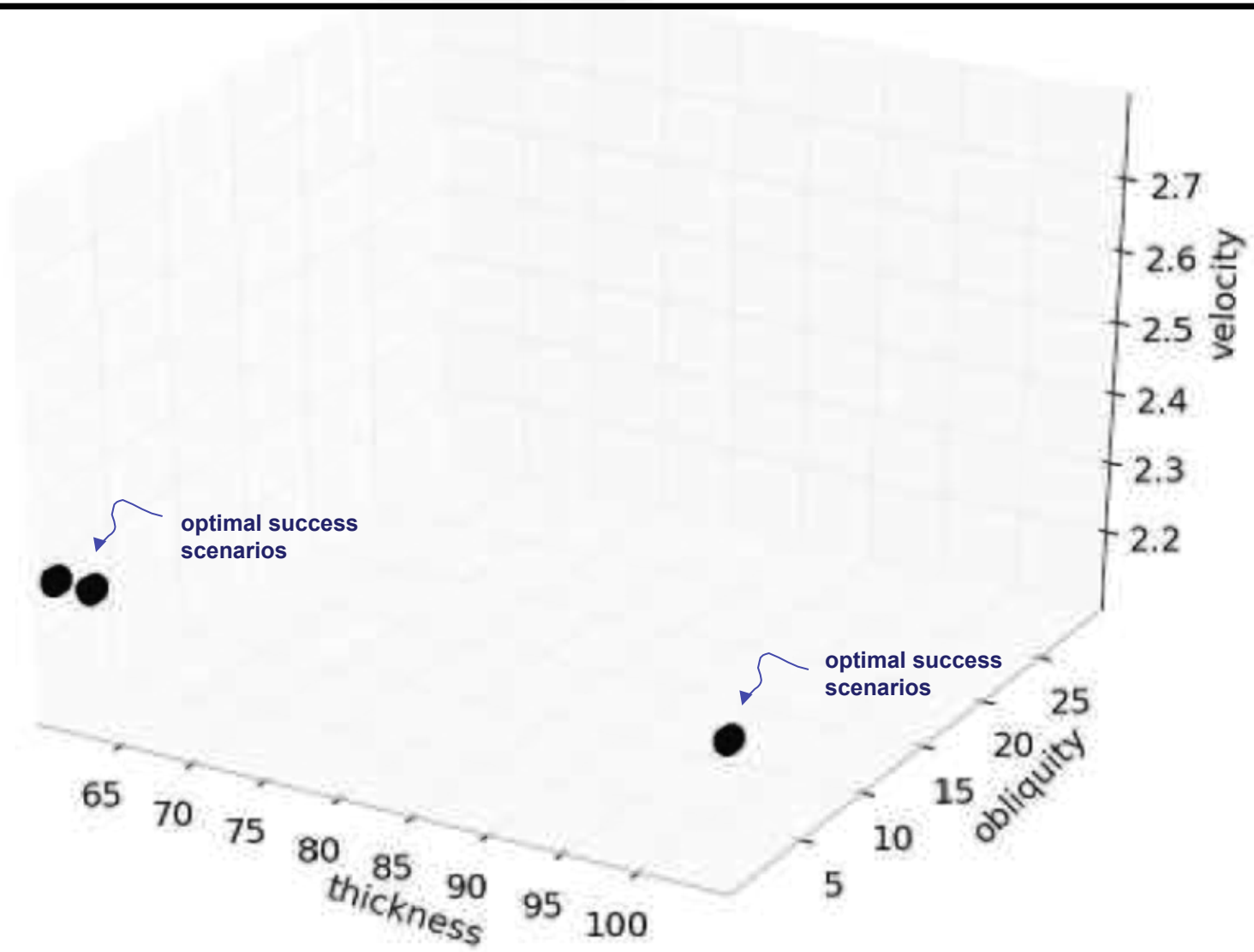
Support Points at iteration 1000

…collapses candidate scenarios…



Support Points at iteration 3000

…solves for extrema in probability of failure



Support Points at iteration 7100

rigorous model validation and engineering design

```
def constraints(rv):
    c = product_measure().load(rv, npts)
    # impose norm on each discrete measure
    for measure in c:
        if not almostEqual(float(measure.mass), 1.0):
            measure.normalize()
    # impose expectation value and other constraints on product measure
    E = float(c.expect(model))
    if E > (target[0] + error[0]) or E < (target[0] - error[0]):
        c.set_expect((target[0],error[0]), model, (x_lb,x_ub), _constraints)
    return c.flatten()  # extract parameter vector of weights and positions

def _constraints(c):
    E = float(c[0].mean)
    if E > (target[1] + error[1]) or E < (target[1] - error[1]):
        c[0].mean = target[1]
    return c

def objective(rv):
    c = product_measure().load(rv, npts)
    return MINMAX * c.pof(failure)
```

$$\mathcal{A} = \left\{ (g,\mu) \;\middle|\; \begin{array}{c} g = \texttt{model} : \mu \in [\texttt{lb},\texttt{ub}] \to \mathbb{R}, \\ \mu = \sum_{i=0}^{3} w_i \delta_i, \\ \sum_{i=0}^{3} w_i = 0, \\ \mathbb{E}_\mu[g] = \texttt{z\_mean}, \\ \bar{\mu} = \texttt{d\_mean} \end{array} \right\}$$

We can test how measurements of new information (by adding a new constraint on the inputs or outputs) alters the probability of failure.

We perform a **design of "experiments"** to discover an information set that can certify the system as "safe" (not failing within the given tolerance)

worst-case bounds on probability of failure

For a viscous Burgers' equation:

$$u_t + uu_x - vu_{xx} = 0$$

with $x \in [-1, 1]$ and viscosity $v > 0$, we have:

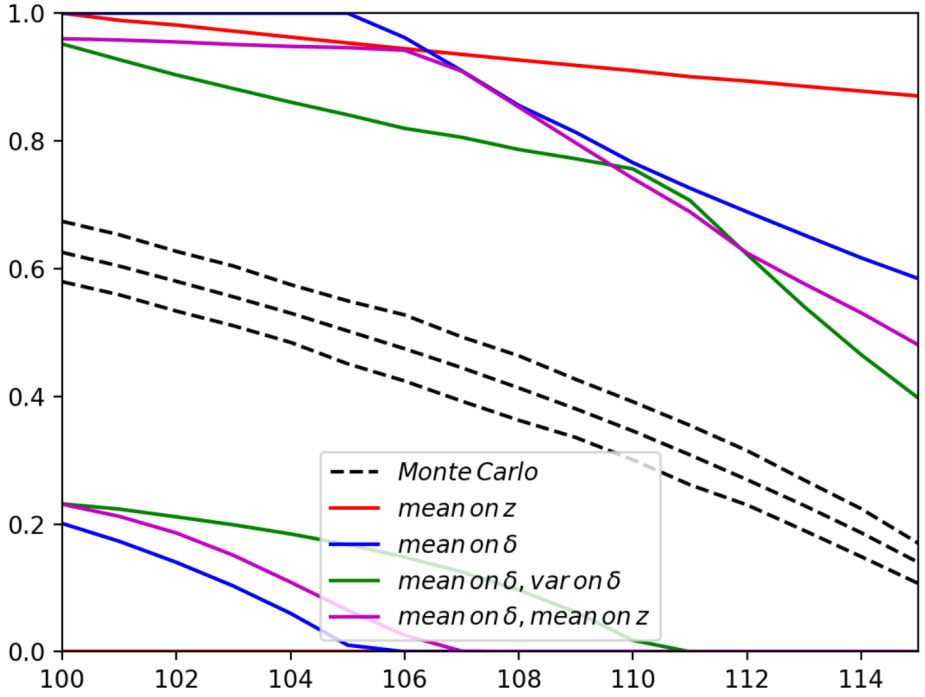
$$u(-1) = 1 + \delta$$
$$u(1) = -1$$

where $\delta > 0$ is a perturbation to the left boundary condition.

OUQ bounds are calculated with a mean constraint on *δ*, a mean constraint and a variance constraint on *δ*, a mean constraint on *z*, and a mean constraint on *δ* and a mean constraint on *z*.

The effect of having different information constraints can be seen on the calculated bounds. More specifically, the presence of additional information can be seen to generally tighten the bounds.

OUQ bounds detect rare events. Compare to bounds calculated with Monte Carlo sampling (100000 points)



Monte Carlo estimate for probability of success

$$P(z > \tfrac{x\bar{z}}{100}) \text{ at } \delta \sim U(0, 0.1)$$

Probability that the shock wave reaches steady state at x% of the mean distance z

outlook and opportunities

- simpler interface for mystic in machine learning
- new high-level optimization workflows and strategies
- new auto-dimensional reduction conditions
- new interpolation strategies and constraints/transforms
- improving speed through multi-grid solvers

- new releases of mystic is available at:
  - https://github.com/uqfoundation

- documentation and tutorials:
  - http://mystic.readthedocs.io
  - https://github.com/mmckerns/tutmom

- I invite any contributors and collaborators:
  - contact me at: mmckerns@uqfoundation.org

End Presentation