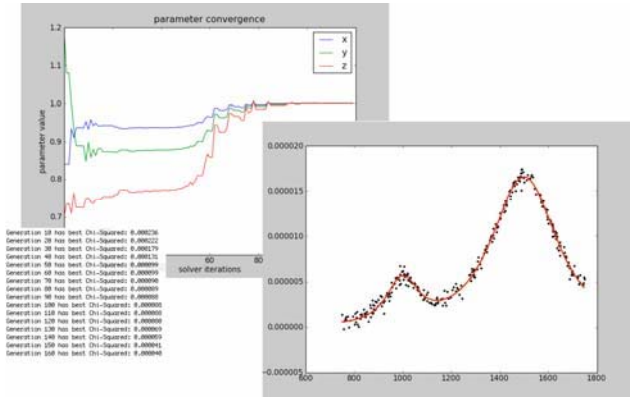


mystic:
a framework for optimization & uncertainty quantification

Mike McKerns

California Institute of Technology

July 1, 2010

http://dev.danse.us/trac/mystic

uncertainty quantification

Discarding information may lead to disaster, whereas over-conservative safety certification may result in economic loss and supplier-client conflict.

Assume that we are interested in certifying that the response function G of a given physical system will not exceed a given safety threshold a with probability at least $1-\epsilon$. This can be formalized as certifying that

$$\mathbb{P}[G(X) \geq a] \leq \epsilon. \tag{2.1}$$

In practice, the event $[G(X) \geq a]$ may represent the crash of an aircraft, the failure of a weapons system, or the average earth tempurature being too high. The measure $\mathbb{P}$ stands for the measure of probability associated with the randomness of some of the input variables of G.

Let $\epsilon \in [0,1]$ denote the *greatest acceptable probability of failure*. We say that the system is *safe* if $\mathbb{P}[G \text{ fails}] \leq \epsilon$ and the system is *unsafe* if $\mathbb{P}[G \text{ fails}] > \epsilon$. By *information*, or a *set of assumptions*, we mean a subset

information alters the probability of failure

Admissible scenarios, $\mathcal{A}$	$\mathcal{U}(\mathcal{A})$	Remarks
$\mathcal{A}$: independence and mean constraints	$\leq 66.4\%$ $= 43.7\%$ $= 37.9\%$	McDiarmid's inequality Optimal McDiarmid *mystic*, H known
$\mathcal{A} \cap \left\{ \mu \,\middle\vert\, \begin{array}{c} \mu\text{-median velocity} \\ = 2.45\,\mathrm{km}\cdot\mathrm{s}^{-1} \end{array} \right\}$	$= 30.0\%$	*mystic*, H known
$\mathcal{A} \cap \{ \mu \,\vert\, \mu\text{-median obliquity} = \frac{\pi}{12} \}$	$= 36.5\%$	*mystic*, H known
$\mathcal{A} \cap \{ \mu \,\vert\, \text{obliquity} = \frac{\pi}{6}\ \mu\text{-a.s.} \}$	$= 28.0\%$	*mystic*, H known
$\mathcal{A}'' := \{\text{uniform measure}\}$	$= 3.8\%$	Exact calculation

Table 6.1: Summary of the upper bounds on the probability of non-perforation for example 6.1 as obtained by various methods.

mystic: a simple model-independent inversion framework

|| User Guide || Download || Tutorials || Manual || License || Feedback ||

About Mystic

The mystic framework provides a collection of optimization algorithms and tools that allows the user to more robustly (and readily) solve optimization problems. All optimization algorithms included in mystic provide workflow at the fitting layer, not just access to the algorithms as function calls. Mystic gives the user fine-grained power to both monitor and steer optimizations as the fit processes are running.

Where possible, mystic optimizers share a common interface, and thus can be easily swapped without the user having to write any new code. Mystic solvers all conform to a solver API, thus also have common method calls to configure and launch an optimization job. For more details, see `mystic.abstract_solver`. The API also makes it easy to bind a favorite 3rd party solver into the mystic framework.

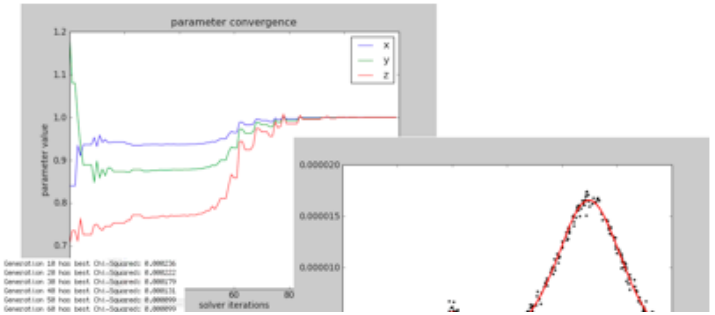
By providing a robust interface designed to allow the user to easily configure and control solvers, mystic reduces the barrier to implementing a target fitting problem as stable code. Thus the user can focus on building their physical models, and not spend time hacking together an interface to optimization code.

Mystic is in the early development stages, and any user feedback is highly appreciated. Contact Mike McKerns [mmckerns at caltech dot edu] with comments, suggestions, and any bugs you may find. A list of known issues is maintained at http://dev.danse.us/trac/mystic/query.

Major Features

Mystic provides a stock set of configurable, controllable solvers with::

- a common interface
- the ability to impose solver-independent bounds constraints
- the ability to apply solver-independent monitors
- the ability to configure solver-independent termination conditions
- a control handler yielding: [pause, continue, exit, and user_callback]
- ease in selecting initial conditions: [initial_guess, random]
- ease in selecting mutation strategies (for differential evolution)



scenarios for cost function analysis

- Definitions

  - F is model; $F(x)$ is model evaluated at some fitted parameter set
  - G is target data; $G(p)$ is series data at some set of system parameters
  - $F(p)$ is model evaluated at series data points

- Optimization

  $cost := min(F(x) - G)^2$

- Parameter Sensitivity

  $cost := - min(F(x) - F(y))^2 \;\; with \; x_j = y_j \; for \; j \neq i$

- Model Validation

  $cost := - min(F(p) - G(p))^2$

measures of parameter sensitivity

- $D_i[F] := sup\{|\, F(x) - F(y) \,||\, x_j = y_j \; for \; i = j\}$

  - the diameter D of a function F measures the model variability over the range of given input parameters

  - diameter evaluations require the solution of a global optimization problem over the range of the inputs (define $cost := D_i[F]^2$)

- $D[F] := (D_1[F]^2 + \ldots + D_N[F]^2)^{1/2}$

  - each independent variable has a sub-diameter D_i which can all be collected to provide a single measure of parameter impact on the model

- $D_i[F] \,/\, D[F]$

  - provides normalized measure of impact of a single parameter

http://www.cacr.caltech.edu/~mmckerns/uq_mystic

measures as data objects

By selecting Diracs as the basis of the 1D measures, the optimizer can discover the weights and positions of the Diracs that maximize $\mu[H = 0]$. For support from any independent random variable x, we can write:

$$\mu_x = \sum_{i=1}^{N_x} w_{x_i} \delta_{x_i}; \text{ where } 1 = \sum_{i=1}^{N_x} w_{x_i}. \tag{6.4}$$

We can also express $\mu[H = 0]$ and $E_\mu[H]$ in terms of Diracs on a 3D product measure:

$$\mu[H = 0] = \sum_{i,j,k} w_{x_i} w_{y_j} w_{z_k} \mathbb{1}[H(x_i, y_j, z_k) = 0] \tag{6.5}$$

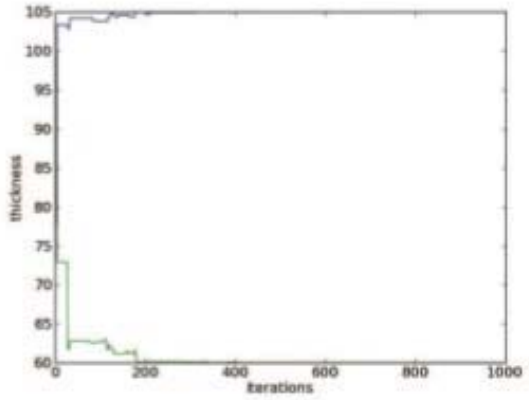
$$E_\mu[H] = \sum_{i,j,k} w_{x_i} w_{y_j} w_{z_k} H(x_i, y_j, z_k) \tag{6.6}$$

apply constraints to measure objects

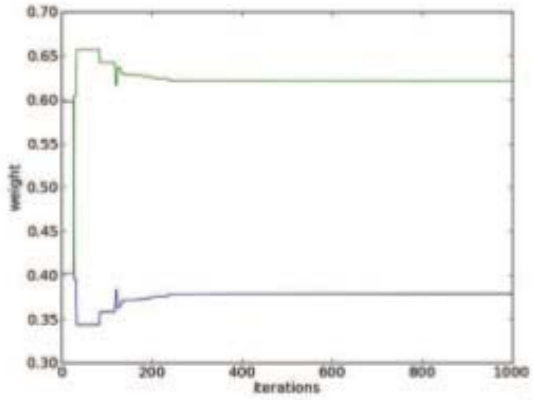
```
x_measure, y_measure, z_measure = unflatten(param)

if sum(x_measure.weights) != 1.0:
    x_measure.normalize()
if sum(y_measure.weights) != 1.0:
    y_measure.normalize()
if sum(z_measure.weights) != 1.0:
    z_measure.normalize()

product_measure = pack((x_measure, y_measure, z_measure))

if product_measure.expect(H) > (m+d) or product_measure.expect(H) < (m-d):
    product_measure.expect = m +/- d

param = flatten(unpack(product_measure))
```

```
param = constrain(param)
cost = costFunction(param)
```

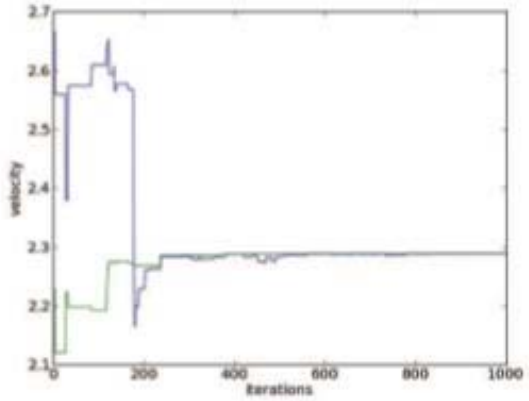
```
param = [wx1, wx2, x1, x2, wy1, wy2, y1, y2, wz1, wz2, z1, z2]
```



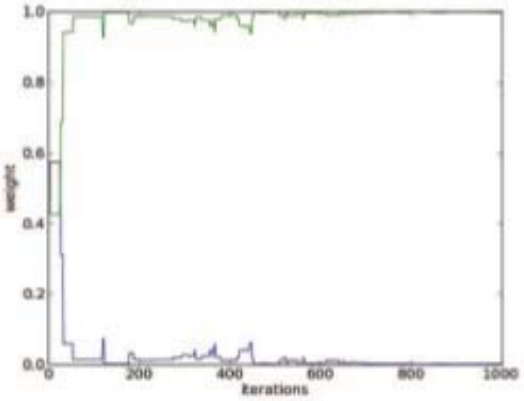
(a) convergence for thickness support



(b) convergence for weight(thickness)

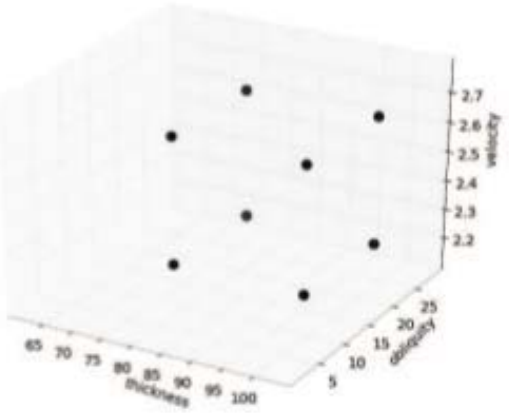


(e) convergence for velocity support

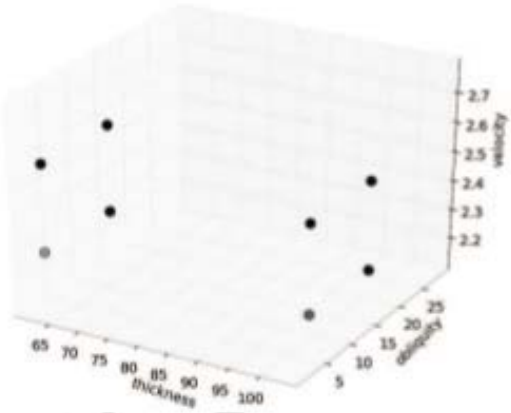


(f) convergence for weight(velocity)

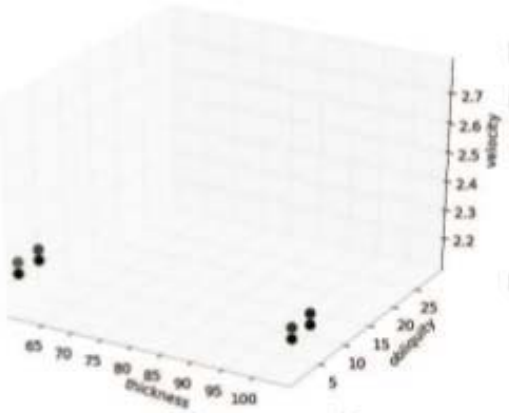
Figure 6.4: Time evolution of the maximizers for Example 6.1 for $\#\text{supp}(\mu_i) \leq 2$ for $i = 1, 2, 3$, as optimized by *mystic*. Thickness quickly converges to the extremes of its range, with weight 0.621 at 60 mils and weight 0.379 at 105 mils. Degeneracy in the obliquity $= 0$ points cause the fluctuations seen in the convergence of weight(obliquity). Velocity converges to a single support point at 2.289 km $\cdot$ s^{-1}, the ballistic limit (Equation 6.2) for thickness $= 105$ mils and obliquity $= 0$.



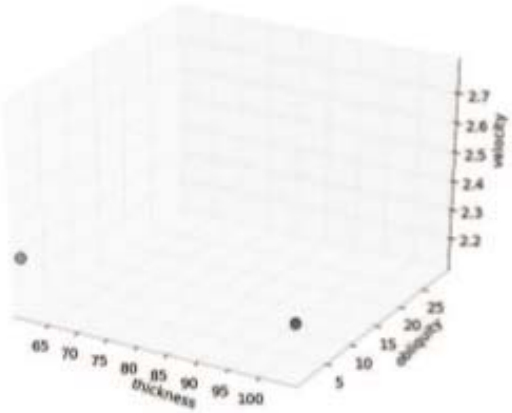
(a) support points at iteration 0



(b) support points at iteration 150



(c) support points at iteration 200



(d) support points at iteration 1000

Figure 6.1: Collapse of maximizers for Example 6.1 to two-point support, for $\#\text{supp}(\mu_i) \leq 2$ for $i = 1, 2, 3$. Velocity and obliquity marginals each collapse to a single Dirac mass, while the plate thickness marginal collapses to have support on the extremes of its range.

End Presentation