

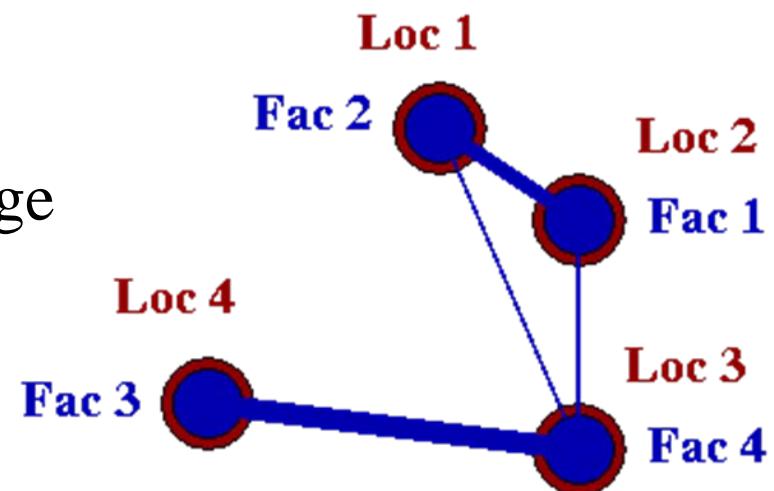


Control Room Accelerator Physics

Day 3
A Primer on the XAL Solver

Outline

1. Overview
2. Solver Quick Start
3. More Advanced Solver Usage
4. Summary



XAL Solver

Overview – Solver of Optimization Problems

The XAL solver expects problems of the basic form

$$\min_{\mathbf{x} \in U} J(\mathbf{x})$$

where

$\mathbf{x} = (x_1, x_2, \dots, x_n)$ are the *parameters* or *variables* of the problem

$U = I_1 \times I_2 \dots \times I_n$; $I_i = [x_{i,\min}, x_{i,\max}]$ are *constraint intervals* for x_i

$J(\mathbf{x})$ is the *objective* for the problem

- Any “problem” must be formulated in terms of
 - Parameters
 - An objective (multiple objects for advanced solver applications)
 - (Evaluations – for advanced solver applications)

XAL Solver

Quick Start

- The easiest way to use the XAL Solver is to...
 1. Creating class that implements **Scorer** interface defining the objective
 1. Represents $J(\mathbf{x})$
 2. Instantiate a **List** of **Variable** objects and their limits
 1. Represents \mathbf{x}
 3. Create a **Problem** object passing the **Variables** and the Scorer to a **ProblemFactory** method.
 1. Represents $\min_{\mathbf{x}} J(\mathbf{x})$
 4. Create a **Solver** object and solve problem

XAL Solver

Quick Start – Defining the Variables

- Instantiate one **Variable** for each parameter

```
from java.util import ArrayList;  
  
from gov.sns.tools.solver import Variable;  
  
arrVars = ArrayList()  
arrVars.add( Variable( "x", 0.0, -25.0, 25.0 ) )  
arrVars.add( Variable( "y", 0.0, -25.0, 25.0 ) )  
arrVars.add( Variable( "z", 0.0, -25.0, 25.0 ) )
```

XAL Solver

Quick Start – The Scorer Interface

- Scorer interface is one method: `score(Trial, List<Variable>)`
 - Trial contains all the state information
 - Second parameter represents \mathbf{x}

```
package gov.sns.tools.solver;

import java.util.List;

/** Score a trial. */
public interface Scorer {
    /**
     * Score the trial.
     * @param trial    the trial to score
     * @return         the score for the trial
     */
    public double score( final Trial trial, final List<Variable> variables );
}
```

XAL Solver

Quick Start – Implementing the Scorer Interface

- Create a class that implements **Scorer**

```
from java.util import List;  
  
from gov.sns.tools.solver import Variable;  
from gov.sns.tools.solver import Solver;  
from gov.sns.tools.solver import Trial;  
from gov.sns.tools.solver import TrialPoint;
```

```
class TargetScorer(Scorer):
```

```
    def __init__( self, mapTargetVal):  
        self._mapTargetVal = mapTargetVal;
```

```
    def score( self, trial, listVar ):  
        norm2 = 0.0;  
        for var in listVar.toArray():  
            diff = trial.getTrialPoint.getValue( var ) - self._mapTargetVal.get( var );  
            norm2 = norm2 + diff*diff;  
        return norm2;
```

Provides a figure of merit evaluation for a given trial set of variable values

mapTargetVal is of type **HashMap**
 $mapTargetVal = \{(var, x_{i,tar})\}$
 $\mathbf{x}_{tar} = (x_{1,tar}, \dots, x_{n,tar})$

$$\text{Score} = \|\mathbf{x} - \mathbf{x}_{tar}\|^2$$

XAL Solver

Quick Start – Instantiation of a Problem Object

- Pick a problem type (min, max, least squares, etc.) by factory method
- Pass Variables and Scorer to ProblemFactory method

```
from java.util import HashMap;  
  
from gov.sns.tools.solver import ProblemFactory;  
  
mapTarget = HashMap();  
mapTarget.put("x", 0.0);  
mapTarget.put("y", 0.0);  
mapTarget.put("z", 0.0);  
  
scorer = TargetScorer(mapTarget);  
  
problem = ProblemFactory.getInverseSquareMinimizerProblem( arrVars, scorer, 0.001 )
```

Numeric tolerance parameter



XAL Solver

Quick Start – Solving the Problem

- Instantiate Solver object with passing a selected Stopper object
 - Pick a Stopper object from the SolveStopperFactory
- Solve the Problem

```
from gov.sns.tools.solver import Solver;
from gov.sns.tools.solver import Stopper;
from gov.sns.tools.solver import SolveStopperFactory;

maxEvalStopper = SolveStopperFactory.maxEvaluationsStopper( 100 );
solver         = Solver( maxEvalStopper );

solver.solve( problem )

# Print out solver statistics
print solver.getScoreBoard()
```

XAL Solver

Quick Start Addendum – Note on “Constraints”

- The XAL Solver also facilitates a weak form of constrained optimization
 - Process works by implementing subclass of **Constraint** base
 - Constraint** objects validate whether a **Trial** object is valid or not

```
/**  
 * Constraint is the class which holds the user's constraint.  
 */  
abstract public class Constraint {  
    protected String name;  
  
    /** Creates a new instance of Constraint  
     * @param aName The name of the constraint.  
     */  
    public Constraint(String aName) { name = aName; }  
  
    /** Used to validate whether a trial is valid.  
     * @param trial The trial to be validated.  
     * @return A trial veto.  
     */  
    abstract public TrialVeto validate(Trial trial);  
}
```

Advanced Use of the XAL Solver

Problems with “Multi-Objectives”

- The XAL Solver may be applied to more general optimization problems
- We consider problems with multi-objectives using satisfaction curves
 - Multi-objectives optimization problems have the form

$$\min_{\mathbf{x} \in U} J(\mathbf{x})$$

where $J(\mathbf{x}) = J_1(\mathbf{x}) + J_2(\mathbf{x}) + \dots + J_n(\mathbf{x})$,

$J_i(\mathbf{x})$ being the *satisfaction* for *objective i*

- Multi-objective Problem's are implemented using the Objective superclass and the Evaluator interface

Advanced Use of the XAL Solver

Advanced Solver: Creating Multi-Objective Problems

- In order to define a **Problem** object...
 - Instantiate one **Variable** object for each free parameter
 - Define each objective
 - Create child class of **Objective** base, implementing the **satisfaction(double)** method (quantifying an objective)
 - The **satisfaction** method must return a **double** between 0 and 1
 - Instantiate an **Objective** child for each problem objective
 - Note: you can also combine all considerations in one objective
 - Create an **Evaluator** object
 - Must implement the **Evaluator** interface which computes the “score” for each objective at the current value of all the variables

Advanced Use of XAL Solver

Declaring Parameters

- Instantiate one **Variable** for each parameter
 - This process is the same as before

```
from java.util import ArrayList;  
  
from gov.sns.tools.solver import Variable;  
  
arrVariables = ArrayList()  
arrVariables.add( Variable( "x", 0.0, -25.0, 25.0 ) )  
arrVariables.add( Variable( "y", 0.0, -25.0, 25.0 ) )
```

Advanced Use of XAL Solver

Defining Objectives

- Create child of **Objective** for each objective type
 - Must implement method **satisfaction(double)**: [0,1]

```
from gov.sns.tools.math.r3 import R3;
from gov.sns.tools.solver import Objective;

class TargetObjective (Objective):

    def __init__( self, name, mapTargetVal):
        Objective.__init__( self, name )
        self._mapTargetVal = mapTargetVal

    def score( self, inputs ):
        norm2 = 0.0;
        for key in inputs.keySet().toArray():
            diff = inputs.get( key ) - self._mapTargetVal.get( key );
            norm2 = norm2 + diff*diff;
        return norm2;

    def satisfaction( self, score ):
        error = self._targetVal - score;
        return 1.0 / ( 1 + error * error )
```

mapTargetVal is of type **HashMap**
represents parameter vector \mathbf{x}

$$\text{Score} = \|\mathbf{x} - \mathbf{x}_{tar}\|^2$$

$$J_i(x_i) = \frac{1}{1 - \|\mathbf{x}_{tar} - \mathbf{x}\|^2}$$

Advanced Use of XAL Solver

Creating the Evaluator

- Create class implementing the Evaluator interface
 - Implement `evaluate(Trial)` for computing objective from parameter values

```
/**  
 * Evaluator is an interface to a custom evaluator for a specific problem.  
 *  
 */  
public interface Evaluator {  
  
    /**  
     * Score the trial.  
     *  
     * @param trial The trial to evaluate.  
     */  
    public void evaluate( Trial trial );  
}
```

Advanced Use of XAL Solver

Creating the Evaluator

- Implementing the Evaluator class
 - Must set the score for each objective: call `Trial.setScore(Objective, Double)`

```
from gov.sns.tools.solver import Evaluator;

class TestEvaluator (Evaluator):
    def __init__( self, objectives, variables ):
        self._objectives = objectives
        self._variables = variables

    def evaluate( self, trial ):
        inputs = HashMap( self._variables.size() )

        for variable in self._variables.toArray():
            value = trial.getTrialPoint().getValue( variable )
            inputs.put( variable, Double( value ) )

        for objective in self._objectives.toArray():
            score = objective.score( inputs )
            trial.setScore( objective, score )
```

Advanced Use of XAL Solver

Creating the Problem

- Instantiate a Problem object with arguments of
 - Objectives
 - Variables
 - Evaluator
- Solve the problem as before

```
mapX1 = HashMap();
mapX1.put("x", 0.0);
mapX1.put("y", 0.0);
mapX1.put("z",0.0);
```

```
mapX2 = HashMap();
mapX2.put("x", 1.0);
mapX2.put("y", 1.0);
mapX2.put("z",1.0);
```

```
J1 = TargetObject(mapX1);
J2 = TargetObject(mapX2);
```

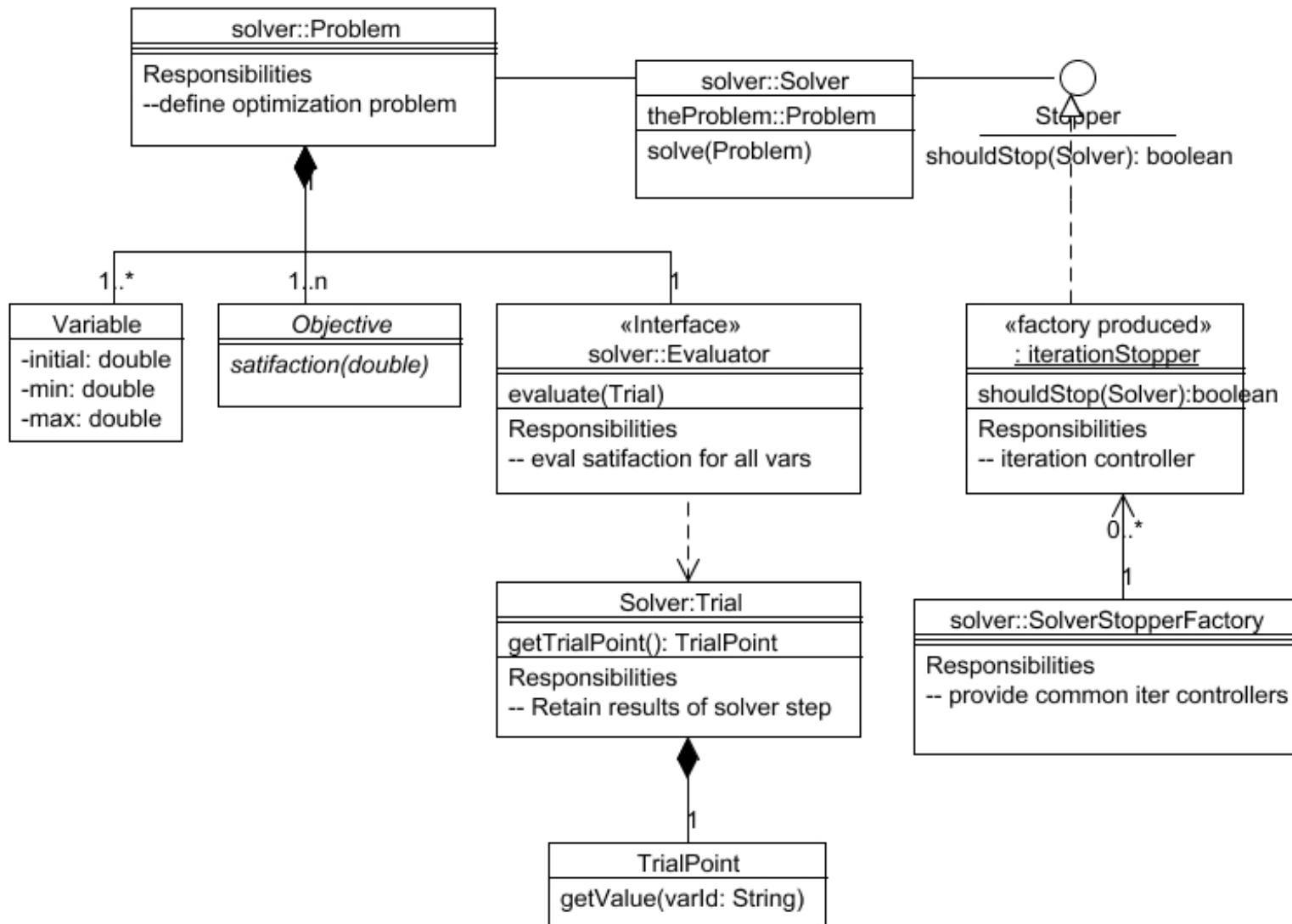
...

```
objectives = ArrayList()
objectives.add( J1 );
objectives.add( J2 );
```

```
evaluator = TestEvaluator( objectives, variables )
```

```
problem = Problem( objectives, variables, evaluator )
```

XAL Solver Architecture



XAL Solver

Summary and Features

- Optimization by search
- Handles noisy data
- Callbacks for monitoring optimization progress
- Satisfaction curves
- Factory for quick configuration of simple problems
- Objectives with dynamic balancing
- Multiple algorithms dynamically adapt to problem