Regression Suite User/Developer Guide

Lakshmi Krishnamurthy

v1.0, 7 June 2012

Introduction

Overview

RegressionSuite is a software test suite that incorporates measurement of the startup lag, measurement of accurate execution times, generating execution statistics, customized input distributions, and processable regression specific details as part of the regular unit tests.

Essentially, RegressionSuite provides the frame-work around which the individual unit regressors are invoked (and details and statistics collected). Unit regressors are grouped into named regressor sets (or modules), and regressors are created by implementing specific regressor interfaces. These two features make regressors particularly amenable to testing analytics (and other similarly state-light) modules.

RegressionSuite Design Objectives and Features

The key objective behind the design of the RegressionSuite is that it enable/ease incorporation of testing to beyond simple unitary unit testing (pass/fail), and allow for (in this edition) clock time based performance measurements. To that end, the following is a comprehensive suite of design goals and features that it handles:

- Execution Time Distribution: A first objective is the measurement of the execution time (clock time, as opposed to CPU time – to gauge the response in a given setting/environment) distribution. Measurement is done by isolating the actual regression run from the preparation and processing (more on this later). Further start-up is treated separately from the routine run, again with a view to measurement

isolation. Central statistical measures such as mean and variance are generated – so are the extremal measures (minimum and maximum execution times).

- Enhanced Unit Testing: The next goal is to enhance the unit testing by extending the typical unit regression tests in a couple of ways. First, RegressionSuite is intended for generating a variety of inputs to span the full range (this can, of course, cause it to be prohibitively expensive for practical use in many situations – strategies for input variance reduction may be employed as suitable – as in identifying the parameter-correlated valid input ranges). Next RegressionSuite also needs to distinguish between situations where the unit tests fail, but the regression is deemed to have succeeded (e.g., over specific inputs). Further RegressionSuite will be extended to capture both the execution time distribution and the unit test success/failure over the given input ranges. As noted earlier, measurement of execution times makes most sense if done by excluding the preparation and output processing times, unit tests need to account for that as well (more on this on the frame-work discussion).
- Pluggable and automated frame-work: RegressionSuite is built using interfaces and frame-works, yet with simple process controls, invocation freedom (minimal constraints), and without side-effects. Both the regression set/modules as well as the unit regressors are built around interfaces, and the core invocation/execution functionality is delegated to the frame-work. The frame-work also generates the regression statistics and other details.
- Regression Details: Details are as elaborate or brief, and their emission is controlled typically at multi-levels – at the frame-work level, at the individual module level, and at the unit regressor level. Field level details are built in by the unit regressor, and there are no limitations on the what constitutes a detail. Being named values, the details are built to be compact/efficient, parseable, and procesesable (XML is not the format of choice). Of course distribution may be generated on any of the number parsable detail fields.

Documentation

Apart from the information provided in this user guide, additional documentation of RegressionSuite functionality and release notes may be found in the RegressionSuite website. Consult the javadoc for elaborate API usage information.

Installation and Dependencies

The core module of RegressionSuite consists of just one jar – RegressionSuite_<<version>>.jar. This contains the complete suite of the entire RegressionSuite library. Download and install this in your class-path.

However, in the attached set of samples, regression is done of the CreditAnalytics library – please refer to the Credit Analytics Installation information (http://code.google.com/p/creditanalytics) for information.

Configuration

No special configuration is required to run the RegressionSuite library. Obviously, configuration may be required for the modules that undergo regression.

In the attached set of samples, regression is done on the CreditAnalytics library – please refer to the Credit Analytics Configuration information (http://code.google.com/p/creditanalytics) for information.

Getting Started

Once you have downloaded, built, and installed the RegressionSuite, you are now ready to execute the RegressionSuite. The best way to start is by running the attached

CreditAnalyticsRegressionEngine sample (make sure you've installed CreditAnalytics in your path to run the latest samples).

The most important calls in the CreditAnalyticsReressionEngine are in its main method. The main initializes the CreditAnalytics library, adds the regressor set, and launches the regression suite. Refer to the javadoc and the comments in the CreditAnalyticsRegressionEngine.java class.

```
	public static void main (final String[] astrArgs) throws Exception {
		CreditAnalyticsRegressionEngine care = new
CreditAnalyticsRegressionEngine (10,
			REGRESSION_DETAIL_MODULE_AGGREGATED);

		/*
		 * Add the regressor sets: Refer to the implementation of the
corresponding regresors
		 */

		care.addRegressorSet (new CreditCurveRegressor());

		care.addRegressorSet (new DiscountCurveRegressor());

		/*
		 * Launch regression - and that's it!
		 */

		care.launch();
	}
```

Regression Suite Class Layout and Package Hierarchy

The full set of RegressionSuite functionality is implemented in just a single package – in addition, it contains an optional samples package that illustrates how to use the framework.. The subsequent sections describe each of the in detail. The layout and the hierarchy of the packages is shown below.

Functional Group	Functional Sub-group / Package / Module	Class
org.drip.regression	core	• RegressionEngine • RegressionRunDetail • RegressionRunOutput • RegressionUtil • RegressorSet • UnitRegressionExecutor • UnitRegressionStat • UnitRegressor
org.drip.regression	sample	• CreditAnalyticsRegressionEngine • DiscountCurveRegressor • FXBasisRegressor • FXCurveRegressor • ZeroCurveRegressor

Package org.drip.regression.core

This package implements the meat of the RegressionSuite functionality.

RegressionEngine

The RegressionEngine class provides the framework and implements the control for the regression engine purposed in the objectives earlier.

Controls can be set in the number of contiguous regression samples generated, as well as detail definition. Currently three levels of details are permitted – decomposed detail at the unit functional level, detail aggregated at the module level, and details aggregated across all modules for a given functional set (with statistics, and pass/fail regression indicators only).

The Regression Engine provides functionality to initialize itself, as well as the ability to add regressor sets/modules. Finally it implements the launch method, which kicks off the entire regression process by executing each unit regressor inside of the regression set module over the specified number of runs, generated the details at the appropriate level, as well as collecting execution time details and processing statistics.

RegressionRunDetail

The RegressionRunDetail class maintains the named field level detailed output of a single unit regression activity.

Individual regressors use the “set” method provided for setting the individual named field during the course of the regression run. The “getFieldMap” returns a field set in the form of a java map.

RegressionRunOutput

The RegressionRunOutput class collects all the output corresponding to a single unit regression activity. It contains the regression function start/stop time stamps, unit execution status and execution time, name of the regression scenario, and the appropriate level of the regression detail.

Each of the above fields is queryable from the instance methods. Finally it also provides a “displayString” method that provides the formatted display of its state.

RegressionUtil

The RegressionUtil implements a set of assorted static utility functionality used by the RegressionSuite.

In this release of RegressionSuite, RegressionUtil provides the ability to compare two doubles to within a specified tolerance. Tolerances can be absolute or relative.

RegressorSet

The RegressorSet interface provides stub functionality for the regression set/module.

It has stubs to get the regression set name, create the set of regressors in the regression set, and to retrieve the regressor set.

UnitRegressionExecutor

The UnitRegressionExecutor abstract class implements the UnitRegressor interface; it also provides the stubs to be implemented by all the unit regressors. It also carries out the actual regression.

The stubs exposed by UnitRegressionExecutor correspond to the different stages of the regression execution – in particular, the pre-, the post-, and the actual regression execution function. The regress function brings it all together – it executes the unit regressor life cycle functions in order, generates the regression outputs at the different stages, and records the time/status.

UnitRegressionStat

The UnitRegressionStat contains all the statistical fields corresponding to a full run of the unit regressor.

In particular, it holds the mean, the minimum, the maximum, the variance, the initialization lag, and the execution time array list. Methods it provides access these fields. The “generateStat” method calculates these statistical measures from a full sequence of runs. Finally, the “displayString” method provides the formatted display of its state.

UnitRegressor

The UnitRegressor interface provides the stubs to be implemented by all the regressors, as well be invoked by the regression engine.

The stubs exposed by UnitRegressor retrieve the regressor name, as well as the regress method that runs and generates the full regression output.

Package org.drip.regression.sample

This package illustrates how to create regression unit tests and customize the regression engine – the test suite chosen here is the Credit Analytics test suite (check out code.google.com/p/creditanalytics).

CreditAnalyticsRegressionEngine

The CreditAnalyticsRegressionEngine customizes the RegressionEngine class for the execution of the CreditAnalytics regression tests.

It sets both the number of samples to be collected, as well as the detail of the individual sample run. CreditAnalyticsRegressionEngine initializes the Credit Analytics library in its initializer, adds the implemented regression sets, and finally launches the whole regression suite.

CreditCurveRegressor, DiscountCurveRegressor, FXCurveRegressor, ZeroCurveRegressor

The individual regressor modules implement the RegressorSet interface that corresponds to their functional modules. These regressors are added to the frame-work for execution by the CreditrAnalyticsRegressionEngine (as discussed earlier).

The RegressorSet implementations expose the set of the UnitRegressors that they've implemented (with the pre-, post-, and execRegression functions, as well as adding the Regression Details), and their names.