SuanShu Introduction

Numerical Method Inc., Jan 2013

Objectives

- SuanShu is a math library of numerical analysis and statistics.
- SuanShu is committed to becoming the most comprehensive and efficient math library, covering the basic as well as the advanced topics in many subjects.
- The SuanShu team is dedicated to providing very high quality and responsive support to the library users.

SuanShu Design

- SuanShu allows easy programming of engineering applications.
- SuanShu is solidly object-oriented.
- SuanShu classes are individually testable.
- SuanShu source code is readable, maintainable, and can be easily modified and extended.

Coverage

- SuanShu is a long term commitment and investment made by Numerical Method Inc.
- We now have a good coverage of the basic numerical analysis algorithms and will cover the advanced topics.

Basic Coverage

- SuanShu now covers most of the basic numerical analysis algorithms (except FFT).
  - numerical differentiation and integration
  - polynomial and Jenkin-Straub
  - root finding
  - unconstrained and constrained optimization for univariate and multivariate functions
  - linear algebra: matrix operations and factorization
  - sparse matrix
  - descriptive statistics
  - random number generators
  - ordinary and partial differential equation solvers
  - time series analysis

SuanShu Statistics Coverage

- Ordinary Least Square (OLS) regression
- Generalized Linear Model (GLM) regression
- a full suite of residual analysis
- Stochastic Differential Equation (SDE) simulation
- a comprehensive library of hypothesis testing:
  - Kolmogorov-Smirnov, D'Agostino, Jarque-Bera, Lilliefors, Shapiro-Wilk, One-way ANOVA, T, Kruskal-Wallis, Siegel-Tukey, Van der Waerden, Wilcoxon rank sum, Wilcoxon signed rank, Breusch-Pagan, ADF, Bartlett, Brown-Forsythe, F, Levene, Pearson's Chi-square, Portmanteau
- time series analysis, univariate and multivariate
  - ARIMA, GARCH modeling, simulation, fitting, and prediction
  - sample and theoretical auto-correlation
- Cointegration
- Kalman filter

Depth (1)

- For each topic SuanShu covers, it goes as deep as possible.
- Take the "numerical integration" package as an example.
  - SuanShu supplies not only the basic Riemann integration procedure.
  - To our knowledge, SuanShu is the only (Java) math library that provides also the various substitution rules to handle improper integrals.

Improper Integral Example

```
/**
 * Student's t distribution.
 */
@Test
public void test_DoubleExponential4RealLine_0020() {
    Gamma gamma = new Gamma();

    final double v = 1;
    double studentT = sqrt(v * PI) * gamma.evaluate(v / 2) / gamma.evaluate((v + 1) / 2);

    double a = Double.NEGATIVE_INFINITY, b = Double.POSITIVE_INFINITY;
    UnivariateRealFunction f = new UnivariateRealFunction() {

        @Override
        public double evaluate(double x) {
            return pow(1 + x * x / v, -(v + 1) / 2);
        }
    };

    EulerMaclaurin integrator = new EulerMaclaurin(3,
            EulerMaclaurin.NewtonCotesType.CLOSED, 1e-15, 5);//only 5 iterations!
    ChangeOfVariable instance =
            new ChangeOfVariable(new DoubleExponential4RealLine(f, 1, a, b), integrator);

    double result = instance.integrate(f, a, b);
    assertEquals(studentT, result, 1e-15);//machine precision
}
```

substitution rule to handle improer integral such as this one

Depth (2)

- Take the Ordinary Least Square regression as an example.
  - SuanShu computes not only the beta estimators and residuals.
  - To our knowledge, SuanShu is the only library that supplements the results with a full suites of residual analysis as some professional statistical software does.
    - beta estimator confidence interval, R-square, adjusted R-square, F, leverage, RSS, TSS, DFFITS, Hadi measure, cook distance, AIC, BIC and etc.

Full Suite of OLS Analysis

```
public void test_OLSRegression_0010() {
    LmProblem problem = new LmProblem(
            new DenseVector(new double[]{2.32, 0.452, 4.53, 12.34, 32.2}),
            new DenseMatrix(new double[][]{
                {1.52, 2.23, 4.31},
                {3.22, 6.34, 3.46},
                {4.32, 12.2, 23.1},
                {10.1034, 43.2, 22.3},
                {12.1, 2.12, 3.27}
            }), true);
    OlsRegression instance = new OlsRegression(problem);
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{3.05526367241960983, -0.34757163218629139, 0.0192186
            instance.beta.betaHat, 1e-15));//summary(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{0.49199823389061365, 0.1945400045869793, 0.299206891
            instance.beta.stderr, 1e-15));//summary(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{6.20991, -1.78663, 0.06423, -1.10080}),
            instance.beta.t, 1e-5));//summary(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{2.7261757160714390, -2.8909172913923262, -0.51439127
            instance.residuals.residuals, 1e-14));//summary(fitted)
    assertEquals(0.9759536870161641398, instance.residuals.R2, 1e-15);//summary(fitted)
    assertEquals(0.903814748064656559, instance.residuals.AR2, 1e-15);//summary(fitted)
    assertEquals(4.036867249675673, instance.residuals.stderr, 1e-14);//summary(fitted)
    assertEquals(13.52880567972044368, instance.residuals.f, 1e-13);//summary(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{-0.4061757160714392, 3.3429172913923262, 5.044391271
            instance.residuals.fitted, 1e-14));//fitted$fitted.value
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{0.5439433910930228, 0.4871594026876104, 0.9837632820
            instance.residuals.leverage, 1e-14));//hatvalues(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{1, -1, -1, 1, 1}),
            instance.residuals.standardized(), 1e-12));//rstandard(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{0, 0, 0, 0, 0}),
            instance.residuals.studentized(), 1e-15));//rstudent(fitted)
    assertEquals(16.2962971915040313, instance.residuals.RSS, 1e-12);//anova(fitted)
    assertEquals(677.7046112, instance.residuals.TSS, 1e-15);//sum(anova(fitted)$"Sum Sq")
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{0, 0, 0, 0, 0}),
            instance.diagnostics.DFFITS, 1e-15));//dffits(fitted)
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{8.546417, 9.160788, 64.654821, 97.259457, 238.187865
            instance.diagnostics.Hadi, 1e-6));
    assertTrue(AreMatrices.equal(new DenseVector(new double[]{0.2981775619898822, 0.2374809079276461, 15.147200438
            instance.diagnostics.cookDistances, 1e-10));//cooks.distance(fitted)
    assertEquals(30.0968853490821, instance.informationCriteria.AIC, 1e-12);//AIC(fitted)
    assertEquals(28.14407491125262, instance.informationCriteria.BIC, 1e-12);//AIC(fitted,k=log(length(y)))
}
```

Efficiency

- SuanShu (from v.1.2) is one of the few (Java) math libraries that support multi-core computing.
- SuanShu is therefore more efficient and is better fit for modern day computing.
- In contrast, most traditional and even the most popular numerical libraries were written for single thread.
  - numerical recipes, netlib, Apache commons math, gsl

Math Libraries in Fortran, C, C++

- Almost all numerical libraries are merely collections of math functions or procedures.
- You cannot easily reuse their code.
  - Even copy & paste would be difficult because of different assumptions.

Object-Oriented Programming

- SuanShu is also a framework of math components so that *you can implement your own math algorithms easily in an object-oriented way.*
- SuanShu is a library of math components, data structures, and reusable utilities that you can put together like Legos to build more complex algorithms.

SuanShu Data Structure

- SuanShu data structures ensure consistency and compatibility by defining and enforcing standards, common objects and interfaces that the whole library and add-on packages use.
  - There is no data structure in most of netlib, gsl. The compatibility of their collections of routines is only coincidental.
- Examples:
  - univariate real function:
  - http://www.numericalmethod.com/javadoc/suanshu/com/numericalmethod/suanshu/analysis/function/rn2r1/univariate/UnivariateRealFunction.html
  - time series:
  - http://www.numericalmethod.com/javadoc/suanshu/com/numericalmethod/suanshu/stats/timeseries/datastructure/package-summary.html
  - distribution function:
  - http://www.numericalmethod.com/javadoc/suanshu/com/numericalmethod/suanshu/stats/distribution/univariate/ProbabilityDistribution.html

Math Functions Componentization

- SuanShu algorithms are coded using modular components which can be
  - reused in your algorithms;
  - assembled together (as in Legos) to build more complex algorithms very easily and quickly (e.g., for quick prototyping);
  - individually tested (unit–testing).

Reusing Householder Reflection (1)

- Householder reflection is one of the operations in QR decomposition. A typical implementation of the QR decomposition looks like this:
  - for (int col = minor+1; col < n; col++) {
    - final double[] qrtCol = qrt[col];
    - double alpha = 0;
    - for (int row = minor; row < m; row++) {
      - alpha -= qrtCol[row] * qrtMinor[row];
    - }
    - alpha /= a * qrtMinor[minor];
    - // Subtract the column vector alpha*v from x.
    - for (int row = minor; row < m; row++) {
      - qrtCol[row] -= alpha * qrtMinor[row];
    - }
  - }
- A similar implementation is found in Apache common math, netlib (http://www.netlib.org/linpack/dqrdc.f), etc.

Reusing Householder Reflection (2)

- One problem with such an implementation is that the code snippet above cannot be reused in other code that also performs the Householder transformation.
  - Copying and pasting the code snippet is not code reuse in the OOP sense.
- In SuanShu design, we made the Householder reflection an independently testable class so that it can be reused in other algorithms.
  - http://www.numericalmethod.com/javadoc/suanshu/com/numericalmethod/suanshu/matrix/doubles/operation/HouseholderReflection.html

SuanShu QR Decomposition

- Here is how we reuse the Householder reflection class in SuanShu implementation of QR decomposition – *simple and clear*.
  - u = concat(new DenseVector(i - 1), u);
  - Hs[i] = new Householder(u);
  - for (int j = i + 1; j <= ncols; ++j) {
    - cols[j] = Hs[i].reflect(cols[j]);
  - }
- The same Householder class is also reused in an OOP way in Bidiagonalization, HessenbergDecomposition, EigenDecomposition, etc.

Modern Programming Paradigm

- SuanShu is written from anew so that it conforms to the modern programming paradigm such as variable naming, code structuring, reusability, readability, maintainability, and software engineering procedure.
- To our knowledge, we believe that we are the first (few) to read the original papers and rewrite them in Java in the modern OOP way.
  - AS 159, AS 288, AS 181, AS 63, AS 109, AS 239, etc.

Comments on AS 159

- The first implementation of Algorithm AS 159 was released in 1981 in FORTRAN.
  - http://lib.stat.cmu.edu/apstat/159
- Looking backward after these 30 years using the modern programming principles, we can make several critiques:
  - The code is very difficult to read and bear little resemblance to the paper.
  - The program structure (flow) is very difficult to follow due to
    - the many level nested loops
    - the too many "GOTO"
    - the too many return code, e.g., IFAULT
  - The variable naming is totally non-informative and is very "encrypted", e.g., IA, IB, IC, ID, IE.
  - There is the lack of enough comments.
  - The code cannot be reused, e.g., the discrete sampling loop could have been made independent and reusable.
- Although there are implementations in C and C++, there are merely translations of the FORTRAN version in different languages, with more or less the same program structure.
  - http://people.sc.fsu.edu/~jburkardt/cpp_src/asa159/asa159.C
  - http://svn.r-project.org/R/trunk/src/appl/rcont.c

SuanShu Implementation of AS 159

- SuanShu rewrites AS 159 from scratch in an OOP way so that it is easy to read, and that modules can be reused.
  - http://www.numericalmethod.com/trac/numericalmethod/browser/public/Misc/AS159.java
- The discrete sampling code is made independent from AS 159 and can be reused in other sampling code.
- The relationships for the 4 sub-matrix sums are made clear.
- The 3 equations in the papers are clearly identified.
- Get rid of the deeply nested loop structures.

Fortran Challenge

- If you think AS 159 is easy to read, try AS 288.

SuanShu Quality

- To ensure very high quality of the code and very few bugs, we have extensive test coverage.
- As of now, SuanShu has a few *thousands* of unit test cases that runs daily.

SuanShu Advantages

- A user, who has little programming experience, can quickly put together SuanShu classes to create solutions for many complex problems.
- A user takes less time to produce more elegant, object-oriented code that is better tuned, has fewer bugs and runs faster.
- Our source code is done in a modern OOP way so that it can be easily used, modified, and extended.

The SuanShu Team

- SuanShu is a professionally created software by the experts in the field.
- The SuanShu team consists of full-time staff and part-time contractors from both computer science and mathematics.
- We are constantly looking for talents to make contributions to the library.
- We list below the core staff in the project.

Professor Haksun Li, CEO

- Haksun has extensive training and experience in computer science and mathematics.
- Haksun had worked in investment banks in various quantitative positions for many years. He created quantitative models for pricing and trading. He also built computer systems for automatic execution and simulation.
- Haksun has a B.S. in Mathematics and a M.S. in Financial Mathematics from the University of Chicago, U.S.A., an M.S., a Ph.D. in Computer Science and Engineering from the University of Michigan, Ann Arbor, U.S.A.

Dr. Ken Yiu, CTO

- Ken is an expert in software design and architecture.
- Ken had worked in an investment bank, where he led a team to design and build an automatic trading system for high frequency trading.
- Ken has a B.Eng., an M.Phil., a Ph.D. in Computer Science and Engineering from the Hong Kong University of Science and Technology.

Dr. Kevin Sun

- Kevin is a seasoned statistician who specializes in applying statistical methods to finance.
- Kevin was a quantitative analyst at an investment bank, where he created mathematical models for quantitative trading.
- Kevin has a B.S. in mathematics, a M.S., in pure mathematics from the University of New South Wales, a M.S., in financial mathematics, and a Ph.D., in statistics from Stanford University.

Professor Chun Yip Yau

- Prof. Chun Yip Yau is an assistant professor in the Statistics department at the Chinese University of Hong Kong.
- Chun Yip created the regression packages and the hypothesis testing package in SuanShu.
- Chun Yip has a B.S. from the University of Hong Kong, a M.Phil. from the Chinese University of Hong Kong, and a Ph.D. in statistics from Columbia University.