More Power (and fun!) in Java Numbers

Frédéric Bapst and Lucy Linder

Haute école d’ingénierie et d’architecture de Fribourg
Pérolles 80, CP 32, CH-1705 Fribourg (Switzerland)
frederic.bapst@hefr.ch

Abstract

We present COJAC (Climbing Over Java Arithmetic Capabilities, https://github.com/Cojac/Cojac), a free tool that lets you boost at runtime the power of Java numbers (float/double)

No source code modification, no recompilation needed

At runtime, COJAC makes floating point numbers able to

• signal anomalies (cancellation, smearing . . . )
• apply interval computation or discrete stochastic arithmetic
• extend the precision to hundreds of significant digits
• convey automatic differentiation
• perform symbolic computation (ongoing work)
• imitate Chebfun (ongoing work)

COJAC can reconcile two points of view

• programmers a priori use native number types (float/double)
• scientific computing has its specific needs and toolboxes

Simple to use — Just tune a JVM option at runtime to make standard numbers powerful (via on-the-fly instrumentation)

Example — Rump polynomial at a “bad” point, naive version

\[ f(x, y) = \frac{1335}{4} y^2 + x^2 (11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11}{2} y^8 + x^2 \]

public class Hello {
    static double pow(double base, int exp) {
        double r = 1.0;
        while (exp > 0) r *= base;
        return r;
    }
    static double myPolyn(double x, double y) {
        return 1335.0 * pow(y, 6) / 4.0;
    }
    public static void main(String[] args) {
        System.out.println(Hello.myPolyn(2.0, 3.0));
    }
}

With another option, you get intervals or discrete stochastic arithmetic (less pessimistic). Now the uncertainty is quantified

\$ java -javaagent:cojac.jar="-Ri" Hello
COJAC: Interval instability [-3.5E+22;3.4E+22]
Hello.myPolyn(Hello.java:12)
-1.1805916207174113E21

1. Standard float and double boosted at runtime

2. Detection of floating point anomalies

To help debugging, COJAC emits warnings when

• a type casting results in precision or information loss
• an operation gives \( \infty \) or NaN
• an addition causes smearing
• there is a strong cancellation effect
• an operation underflows
• two very close numbers are compared
• an integer overflow occurs

\$ java -javaagent:cojac.jar Hello
COJAC: Smearing: DSUB Hello.myPolyn(Hello.java:13)
COJAC: Cancellation: DADD Hello.myPolyn(Hello.java:12)

3. Interval computation

With another option, you get intervals or discrete stochastic arithmetic (less pessimistic). Now the uncertainty is quantified

\$ java -javaagent:cojac.jar="-Ri" Hello
COJAC: Interval instability [-3.5E+22;3.4E+22]
Hello.myPolyn(Hello.java:12)
-1.1805916207174113E21

\$ java -javaagent:cojac.jar="-Ri" Hello
COJAC: Interval instability [-3.5E+22;3.4E+22]
Hello.myPolyn(Hello.java:12)
-1.1805916207174113E21

4. Higher precision

Maybe you want BigDecimal instead of double, to have full control over the underlying precision (here 40 significant digits)

\$ java -javaagent:cojac.jar="-Rb 40" Hello
f'("x",3): -38954.0
Correct result!

5. Automatic differentiation

One additional line automatically brings the derivative in \( x \). This is automatic differentiation, a technique completely different from both numerical and symbolic differentiation

public static void main(String[] args) {
    double r, x = 2.0, y = 3.0;
    r = COJAC_MAGIC_asDerivationTarget(x);
    System.out.println("f\((x,3)\): \"", r);
    System.out.println("f'\((x,3)\): \"", COJAC_derivative(r));
}

\$ java -javaagent:cojac.jar="-Rb 40" Hello
f\((x,3)\): 238845.58333333334
f'\((x,3)\): -38954.0

6. Symbolic processing and Chebfun (ongoing work)

We are able to store the whole symbolic expression tree of a result, a first step towards symbolic processing (finding roots, computing integrals . . . ). Numbers become functions

public static void main(String[] args) {
    double x = COJAC_MAGIC_identity();
    double r = myPolyn(x, y);
    System.out.println(COJAC_toString(r));
}

\$ java -javaagent:cojac.jar="-Ry" Hello
r(2): 238845.58333333334
r(\(x\)) = \text{ADD}(\ldots, \text{ADD}(\text{MUL}(x, \ldots(\text{SUB}(\text{MUL}(11.0, \text{MUL}(x(\ldots)
file: 238845.58333333334

We’ll try to port Matlab’s Chebfun and use Chebyshev interpolation to find roots efficiently and accurately

Conclusion — we need you!

COJAC gives super powers to float/double . . . Is it useful?

Dear colloquium participants, please give us feedback

https://github.com/Cojac/Cojac
frederic.bapst@hefr.ch

Colloque Numérique Suisse / Schweizer Numerik Kolloquium, Université de Fribourg, April 22, 2016