

2.2 Libraries and Clients

Library. A module whose methods are primarily intended for use by many other programs.

Client. Program that calls a library.

API. Contract between client and implementation.

Implementation. Program that implements the methods in an API.

Random Numbers

The generation of random numbers is far too important to leave to chance. Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. 

Jon von Neumann (left), ENIAC (right)

Random Numbers

random.

Our library to generate pseudo-random numbers.

public class StdRandom

// between a and b
public static double uniform(double a, double b) {
    return a + Math.random() * (b - a);
}

// between 0 and N - 1
public static int uniform(int N) {
    return (int) (Math.random() * N);
}

// true with probability p
public static boolean bernoulli(double p) {
    return Math.random() < p;
}

// gaussian with mean = 0, stddev = 1
public static double gaussian() {
    // see Exercise 1.2.27
}

// gaussian with given mean and stddev
public static double gaussian(double mean, double stddev) {
    return mean + (stddev * gaussian());
}

...
public class StdRandom {
    ...
    public static void max(String[] args) {
        int N = Integer.parseInt(args[0]);
        for (int i = 0; i < N; i++) {
            StdOut.printf("%2d ", uniform(100));
            StdOut.printf("%8.5f ", uniform(10.0, 99.0));
            StdOut.printf("%5b ", bernoulli(.5));
            StdOut.printf("%7.5f ", gaussian(9.0, 2.0));
            StdOut.println();
        }
    }
}

java StdRandom 5
61 21.76541  true 9.30910
57 43.64327 false 9.42369
31 30.86201  true 9.06366
92 39.59314  true 9.00896
36 28.27256 false 8.66800

StdStats

public class StdStats {
    public static double max(double[] a) {
        double max = Double.NEGATIVE_INFINITY;
        for (int i = 0; i < a.length; i++)
            if (a[i] > max) max = a[i];
        return max;
    }

    public static double mean(double[] a) {
        double sum = 0.0;
        for (int i = 0; i < a.length; i++)
            sum = sum + a[i];
        return sum / a.length;
    }

    public static double stddev(double[] a) {
        // see text
    }

    public static double median(double[] a) {
        // see text
    }
}

java StdStats 5
61 21.76541  true 9.30910
57 43.64327 false 9.42369
31 30.86201  true 9.06366
92 39.59314  true 9.00896
36 28.27256 false 8.66800

Standard Statistics

Ex. Library to compute statistics on an array of real numbers.

public class StdStats {
    double max(double[] a) { return Double.MAX_VALUE; }
    double min(double[] a) { return Double.MIN_VALUE; }
    double mean(double[] a) { return 0.0; }
    double var(double[] a) { return 0.0; }
    double stddev(double[] a) { return 0.0; }
    double median(double[] a) { return 0.0; }
    void plotPoints(double[] a) { return; }
    void plotLines(double[] a) { return; }
    void plotBars(double[] a) { return; }
}

\[ \mu = \frac{a_1 + a_2 + \cdots + a_n}{n} \]
\[ \sigma^2 = \frac{(a_1 - \mu)^2 + (a_2 - \mu)^2 + \cdots + (a_n - \mu)^2}{n-1} \]

Modular Programming

Ex. Library to compute statistics on an array of real numbers.

public class StdStats {
    double max(double[] a) { return Double.MAX_VALUE; }
    double min(double[] a) { return Double.MIN_VALUE; }
    double mean(double[] a) { return 0.0; }
    double var(double[] a) { return 0.0; }
    double stddev(double[] a) { return 0.0; }
    double median(double[] a) { return 0.0; }
    void plotPoints(double[] a) { return; }
    void plotLines(double[] a) { return; }
    void plotBars(double[] a) { return; }
}

\[ \mu = \frac{a_1 + a_2 + \cdots + a_n}{n} \]
\[ \sigma^2 = \frac{(a_1 - \mu)^2 + (a_2 - \mu)^2 + \cdots + (a_n - \mu)^2}{n-1} \]

Unit Testing

Unit test. Include main() to test each library.
**Modular Programming**

- Divide program into self-contained pieces.
- Test each piece individually.
- Combine pieces to make program.

Ex. Flip N coins. How many heads?
- Read arguments from user.
- Flip one fair coin.
- Flip N fair coins and count number of heads.
- Repeat simulation, counting number of times each outcome occurs.
- Compare with theoretical predictions.

```java
public class Bernoulli {
    public static int binomial(int N) {
        int heads = 0;
        for (int j = 0; j < N; j++)
            if (StdRandom.bernoulli(0.5)) heads++;
        return heads;
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int T = Integer.parseInt(args[1]);
        int[] freq = new int[N+1];
        for (int i = 0; i < T; i++)
            freq[binomial(N)]++;
        double[] normalized = new double[N+1];
        for (int i = 0; i <= N; i++)
            normalized[i] = (double) freq[i] / T;
        StdStats.plotBars(normalized);
        double mean = N / 2.0, stddev = Math.sqrt(N) / 2.0;
        double[] phi = new double[N+1];
        for (int i = 0; i <= N; i++)
            phi[i] = Gaussian.phi(i, mean, stddev);
        StdStats.plotLines(phi);
    }
}
```

**Bernoulli Trials**

- Read arguments from user.
- Flip one fair coin.
- Flip N fair coins and count number of heads.
- Repeat simulation, counting number of times each outcome occurs.
- Compare with theoretical predictions.

**Dependency Graph**

- Build relatively complicated program by combining several small, independent, modules.

**Libraries**

- Makes code easier to understand.
- Makes code easier to debug.
- Makes code easier to maintain and improve.
- Makes code easier to reuse.

**Discrete Distribution**

- Given an array of weights (that sum to 1), choose an index at random with probability equal to its weight.

```java
public static int discrete(double[] p) {
    // check that weights are nonnegative and sum to 1
    double c = Math.random();
    double sum = 0.0;
    for (int i = 0; i < p.length; i++)
        if (sum >= c) return i;
    return -1;  // something went wrong
}
```

Extra Slides