4. 乱数とヒストグラム

2015/11/12

1　準備

講義で説明した一様乱数のヒストグラムを作成する準備として以下の作業をします。

1. プロジェクト Random を作成する。
2. プロジェクト Random 内にパッケージ histogram を作成する。
3. プロジェクト Random 内にパッケージ samples を生成する。
4. プロジェクト Random 内のライブラリとして、微分方程式の際に使用した MyLib を登録する。

2　乱数

一様乱数とはある範囲のなかで、一様な頻度ででたらめな値を生成するものです。Java の場合、Math.random() によって、区間 $[0, 1)$ の乱数を生成することができます。

別紙のクラス Uniform は、区間 $[a, b)$ の一様乱数を生成するクラスです。コンストラクタでは、区間を指定します。メソッド getNext() では、Math.random() を用いて、区間 $[a, b)$ の一様乱数を生成しています。なぜ、生成できるかを考え、簡易報告に記載してください。また、このプログラムをパッケージ samples に作成しましょう。

別紙のクラス Uniform の main() メソッドでは、区間 $[0, 1)$ の一様乱数を 100,000 個生成するようになっています。該当する場所を見つけましょう。

3　ヒストグラム

3.1　プログラムの完成

ヒストグラム (histogram) とは、事象毎の頻度を表すものです。ある現象が区間 $[a, b)$ の間の実数値として出現する場合には、各値の出現頻度を調べることは意味がありません。あるいは、その現象が非常に広い区間 $[a, b)$、例えば $[0, 100, 000)$ の間の整数値として出現する場合でも、各値の出現頻度を調べることは、そのサンプル数が 100,000 に比べて桁違いに多い場合でなければ、意味がありません。そのような場合には、区間を小区間に分けて、各小区間内の値が出現する頻度を調べることが有効です。小区間を bin と呼びます。

別紙のヒストグラムのクラス `Histogram` では、コンストラクタに区間 $[a, b)$ と、それを小区間に等分するための整数が与え、初期化します。コンストラクタ内では、頻度を数える配列 `hist` を生成しています。

このプログラムをパッケージ `histogram` に作成しましょう。また、与えられた値 x がはいる bin の番号を k として、その bin の頻度を一つ増やすメソッド `put` の内容を完成させましょう。

3.2　ヒストグラムの図示

クラス `Uniform` の `main()` メソッドでは、区間 $[0, 1)$ の一様乱数を 100,000 個生成するようになっています。その結果は、`output.txt` ファイルに出力されます。出力ファイルには、区間の中央値とその区間に入った数値の相対頻度が、全区間の頻度と区間幅の積の和が 1 になるように記述されます。これを Program 3.1 を用いて図示しましょう。

```
set terminal png enhanced
set xlabel "x"
set ylabel "p"
set xrange [0:4]
set yrange [0:1.5]
set ytic 0.2
set output "uniform.png"
set title "Uniform Distribution"
set style fill solid
plot "output.txt" with boxes notitle
```

Program 3.1: `histogram.plt`

また、生成する乱数の範囲を $[-0.3, 0.7)$ として、正しく乱数が生成され、ヒストグラムが図示できることを確認しましょう。

3.3 (発展) 平均と分散

余力のある人は、ヒストグラムの高さの平均と標準偏差を数値的に求めるように、クラス Histgram を拡張しましょう。また、講義で説明したように、平均と標準偏差は手で計算することができます。数値的計算と比較しましょう。

Uniform.java

```
package samples;

import histogram.Histogram;
import java.awt.geom.Point2D;
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.List;
import myLib.utils.FileIO;

/**
 * 一様乱数の例
 *
 * @author tadaki
 */
public class Uniform {

    private final double a;
    private final double b;

    /**
     * コンストラクタ
     *
     * @param a 下限
     * @param b 上限
     */
    public Uniform(double a, double b) {
        this.a = a;
        this.b = b;
    }

    /**
     * 乱数を一つ生成
     * @return
     */
    public double getNext() {
        return (b - a) * Math.random() + a;
    }

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    public static void main(String[] args) throws IOException {
        double a = 0.;//下限
        double b = 1.;//上限
        int m = 100;//binの数
```

Uniform.java

```
        int n = 100000;//乱数の総数
        //ヒストグラムを生成
        Histogram histogram = new Histogram(a, b, m);
        //乱数をn個生成し、ヒストグラムへ登録
        Uniform uniform = new Uniform(a, b);
        for (int i = 0; i < n; i++) {
            double x = uniform.getNext();
            histogram.put(x);
        }
        //ヒストグラムを出力
        List<Point2D.Double> plist = histogram.getHistogram();
        try (BufferedWriter out = FileIO.openWriter("output.txt")) {
            for (Point2D.Double p : plist) {
                FileIO.writeSSV(out, p.x, p.y);
            }
        }
    }

}
```

Histogram.java

```java
package histogram;

import java.awt.geom.Point2D;
import java.util.List;
import myLib.utils.Utils;

/**
 * ヒストグラム
 *
 * @author tadaki
 */
public class Histogram {

    private final double a;//範囲の下限
    private final double b;//範囲の上限
    private final double w;//bin の幅
    private int hist[];//ヒストグラム

    /**
     * コンストラクタ
     *
     * @param a 下限
     * @param b 上限
     * @param M binの数
     */
    public Histogram(double a, double b, int M) {
        this.a = a;
        this.b = b;
        w = (b - a) / M;
        hist = new int[M];
    }

    /**
     * 値を一つ登録する
     *
     * @param x
     * @return
     */
    public boolean put(double x) {
        if (x < a || x >= b) {//範囲外の場合
            return false;
        }
        //xが入るべきbinの番号を調べる
        int k =
        //bin のカウントを一つ増やす
        hist[k]++;
```

```
        return true;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public int[] getHist() {
        return hist;
    }

    /**
     * 結果をリストとして取得する
     *
     * 値は確率になるように規格化する
     *
     * @return
     */
    public List<Point2D.Double> getHistogram() {
        List<Point2D.Double> plist = Utils.createList();
        //カウントの総和を求める
        int sum = 0;
        for (int i = 0; i < hist.length; i++) {
            sum += hist[i];
        }

        for (int i = 0; i < hist.length; i++) {
            double x = a + i * w + w / 2.;//binの中央値
            double y = (double) hist[i] / sum / w;//binに入る割合
            plist.add(new Point2D.Double(x, y));
        }
        return plist;
    }
}
```