Лабораторная работа №6

Выполнил: Магера Никита Алексеевич

Студент группы 6203-010302D

Ход выполнения

Задание 1

В 1 задании необходимо в классе FunctionPoint добавить метод Integral, реализующий численное интегрирование методом трапеций. Метод должен принимать функцию, границы интегрирования и шаг дискретизации, проверять корректность интервала и вычислять интеграл. Затем протестировать метод на функции e^x от 0 до 1, определить шаг для точности 7 знаков после запятой. Теоретическое значение: e-1 ≈ 1.718281828. Экспериментально установлено, что для достижения точности в 7 знаке после запятой требуется шаг дискретизации порядка 0.001. (Рис.1-2)

```
public static double Integral(Function function, double left, double right, double step) {  1 usage  new *
    // не выходим ли за границы области определения
    if (left < function.getLeftDomainBorder() || right > function.getRightDomainBorder()) {
        throw new IllegalArgumentException("Интервал интегрирования выходит за границы области определения функции");
    }
    // вычисление интеграла методом трапеций
    double integral = 0.0;
    double need_x = left;
    double need_y = function.getFunctionValue(need_x);
    // идем пока не дойдем до правой границы
    while (need_x < right) {
        // следующая точка
        double next_x = need_x + step;
        if (next_x > right) {
            next_x = right;
        }
        // значение функции в следующей точке
        double next_y = function.getFunctionValue(next_x);
        // площадь трапеции: (основание1 + основание2) * высота / 2
        double trapezoid = (need_y + next_y) * (next_x - need_x) / 2.0;
        integral += trapezoid;
        // переходим к следующему отрезку
        need_x = next_x;
        need_y = next_y;
    }
    return integral;
}
```

Рис. 1

```
проверка интегрирования
теоретическое значение ∫e^x dx от 0 до 1 = 1.718281828459045
Шаг: 1.0  Результат: 1.8591409142295225, Ошибка: 0.14085908577047745
не удовлетворяет условию
Шаг: 0.1  Результат: 1.7197134913893144, Ошибка: 0.0014316629302693062
не удовлетворяет условию
Шаг: 0.01  Результат: 1.7182961474504181, Ошибка: 1.431899137305237E-5
не удовлетворяет условию
Шаг: 0.001  Результат: 1.7182819716491948, Ошибка: 1.4319014973729338E-7
не удовлетворяет условию
Шаг: 1.0E-4  Результат: 1.7182818298909435, Ошибка: 1.4318983776462346E-9
7 знаков, результат удовлетворяет условию
```

Рис. 2

Задание 2

В 2 задании нужно создать класс `Task` в пакете `threads`, который будет хранить параметры задания для вычисления интеграла. Класс должен содержать поля для функции (логарифм), левой и правой границ интегрирования, шага и результата вычислений. Функция-логарифм генерируется со случайным основанием от 1 до 10, левая граница — от 0 до 100, правая граница — от 100 до 200, шаг — от 0 до 1. Затем реализуется метод `nonThread()`, который последовательно выполняет 100 таких заданий. Для каждого задания метод вычисляет определенный интеграл функции логарифма на заданном интервале с указанным шагом, после чего выводит параметры задания в формате "Source <левая граница> <правая граница> <шаг>" и результат в формате "Result <левая граница> <правая граница> <шаг> <значение интеграла>". Все вычисления выполняются в одном потоке, что позволяет оценить базовое время выполнения без использования многопоточности.(Рис.3-4)

```
// последовательная версия программы
public static void nonThread() {  1 usage  new *
    //объект Task с 100 заданиями
    Task task = new Task( tasksCount: 100);
    for (int i = 0; i < task.getTasksCount(); i++) {
        // логарифм со случайным основанием от 1 до 10
        double log = 1 + Math.random() * 9;
        task.setFunction(new functions.basic.Log(log));
        // левая граница: 0 - 100
        task.setLeftX(Math.random() * 100);
        // правая граница: 100 - 200
        task.setRightX(100 + Math.random() * 100);
        // шаг 0 1
        task.setStep(Math.random());
        // вывод
        System.out.println("Source " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep());
        try {
            // вычисление интеграла
            double integral = Functions.Integral(task.getFunction(), task.getLeftX(), task.getRightX(), task.getStep());
            System.out.println("Result " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep() + " " + integral);

        } catch (Exception e) {
            System.out.println("ошибка интегрирования: " + e.getMessage());
        }
    }
    System.out.println("Выполнено " + task.getTasksCount() + " заданий.");
}
```

Рис.3

```
Result 51.33388442711282 180.4440588746545 0.2844056267499938 304.363069499722
Source 35.56841401314376 174.83462554375973 0.617394413035064
Result 35.56841401314376 174.83462554375973 0.617394413035064 300.8552666627543
Source 2.4219427518801018 190.25802920321667 0.7639419325006767
Result 2.4219427518801018 190.25802920321667 0.7639419325006767 660.9971759185038
Source 91.83043227299463 181.06590386972258 0.9330951896610484
Result 91.83043227299463 181.06590386972258 0.9330951896610484 222.93885799715076
Source 99.88633324825496 104.56632722269293 0.3164963210514423
Result 99.88633324825496 104.56632722269293 0.3164963210514423 9.98146125750764
Source 45.7593116834891 183.47287702886473 0.04314310321210768
Result 45.7593116834891 183.47287702886473 0.04314310321210768 360.1503199377836
Source 56.11658541290217 104.8878910329398 0.7709532539707321
Result 56.11658541290217 104.8878910329398 0.7709532539707321 921.0879642772388
Source 13.530342954668518 144.81226771025308 0.5601823502081682
Result 13.530342954668518 144.81226771025308 0.5601823502081682 260.7457640154454
Source 27.176062608031813 197.8967335598192 0.43253434029872395
Result 27.176062608031813 197.8967335598192 0.43253434029872395 442.52515231843915
Source 49.06694740782002 160.9972431231568 0.8098433418193903
Result 49.06694740782002 160.9972431231568 0.8098433418193903 619.4697825415116
Source 77.13617110464482 166.06976070185468 0.8474933752420433
Result 77.13617110464482 166.06976070185468 0.8474933752420433 524.7554883377562
Source 57.66993242790125 183.03735709513796 0.530282706856159
Result 57.66993242790125 183.03735709513796 0.530282706856159 394.62951386288927
Source 23.910809062809857 116.48524671039686 0.2822280871463688
Result 23.910809062809857 116.48524671039686 0.2822280871463688 186.93745760800073
Source 85.09228355834763 101.38695004384036 0.7568431475074328
Result 85.09228355834763 101.38695004384036 0.7568431475074328 45.86242962261023
Source 10.731668696844988 162.53756035369915 0.4207146181235708
Result 10.731668696844988 162.53756035369915 0.4207146181235708 596.216626093298
Source 38.70015659128998 139.96760428443264 0.5207390037033783
Result 38.70015659128998 139.96760428443264 0.5207390037033783 204.27746729173015
Source 72.4050529295221 198.33376708027146 0.03417085684227317
Result 72.4050529295221 198.33376708027146 0.03417085684227317 727.3809385953401
Source 22.360667813630243 132.13031360195075 0.24067407115692263
Result 22.360667813630243 132.13031360195075 0.24067407115692263 243.25308037905492
Source 90.49081009741266 150.5377581013842 0.6268551480473792
Result 90.49081009741266 150.5377581013842 0.6268551480473792 145.3164772012388
Source 91.69711372450631 175.8229638054374 0.9362977897304738
Result 91.69711372450631 175.8229638054374 0.9362977897304738 1847.4492192167158
Выполнено 100 заданий.
```

Рис.4

Задание 3

В третьем задании должны быть созданы классы SimpleGenerator и SimpleIntegrator, реализующие интерфейс Runnable. SimpleGenerator формирует задания в цикле, занося параметры в объект Task и выводя сообщения "Source". SimpleIntegrator в цикле извлекает данные из Task, вычисляет интегралы через Functions.Integral и выводит "Result". При тестировании выявлены проблемы многопоточности: NullPointerException возникал при попытке интегратора получить данные до их генерации, также наблюдалось смешивание параметров из разных заданий. Для устранения добавлена проверка на null с ожиданием и блоки синхронизации synchronized(task) в методах run(). Проведены эксперименты с приоритетами потоков: при высоком приоритете генератора задания создавались быстрее, чем обрабатывались; при высоком приоритете интегратора учащались попытки чтения неготовых данных. После синхронизации исключения исчезли, данные передаются целостно, потоки корректно завершают все 100 заданий.(Рис.5-7)

```java
public class SimpleIntegrator implements Runnable {  1 usage  new *
    private Task task;  11 usages
    public SimpleIntegrator(Task task) { this.task = task; }

    public void run() {  new *
        for (int i = 0; i < task.getTasksCount(); i++) {
            // synchronized
            synchronized (task) {
                try {
                    // ждем пока генератор создаст задание
                    while (task.getFunction() == null) {
                        task.wait();
                    }

                    // берем текущие данные
                    double leftX = task.getLeftX();
                    double rightX = task.getRightX();
                    double step = task.getStep();

                    // вычисляем интеграл
                    double integral = Functions.Integral(task.getFunction(), leftX, rightX, step);
                    System.out.println("Result " + leftX + " " + rightX + " " + step + " " + integral);

                    // очищаем
                    task.setFunction(null);
                    task.notifyAll();

                } catch (Exception e) {
                    System.out.println("Integrator error: " + e.getMessage());
                }
            }

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {
                break;
            }
        }
    }
}
```

Рис.5

```java
public class SimpleIntegrator implements Runnable {  1 usage  new *
    private Task task;  11 usages
    public SimpleIntegrator(Task task) { this.task = task; }

    public void run() {  new *
        for (int i = 0; i < task.getTasksCount(); i++) {
            // synchronized
            synchronized (task) {
                try {
                    // ждем пока генератор создаст задание
                    while (task.getFunction() == null) {
                        task.wait();
                    }

                    // берем текущие данные
                    double leftX = task.getLeftX();
                    double rightX = task.getRightX();
                    double step = task.getStep();

                    // вычисляем интеграл
                    double integral = Functions.Integral(task.getFunction(), leftX, rightX, step);

                    // выводим результат
                    System.out.println("Result " + leftX + " " + rightX + " " + step + " " + integral);

                    // очищаем и будим генератор
                    task.setFunction(null);
                    task.notifyAll();

                } catch (Exception e) {
                    System.out.println("Integrator error: " + e.getMessage());
                }
            }

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {
                break;
            }
        }
    }
}
```

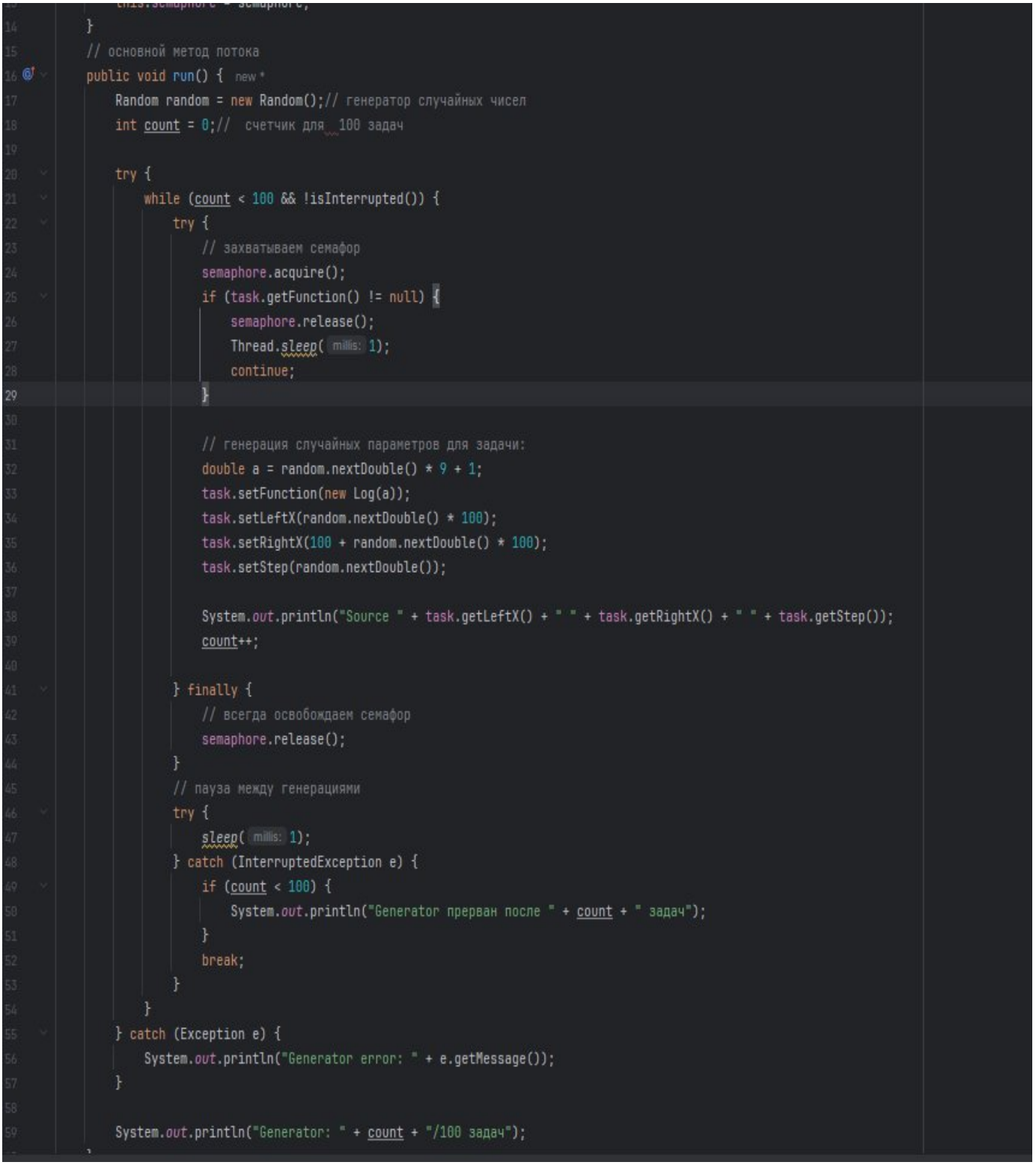
Рис.6

```
Result 82.43875183750514 160.05928896748236 0.29134644477034577 162.4163014392222
Source 94.37545182206102 155.00650591795585 0.8348814746976767
Result 94.37545182206102 155.00650591795585 0.8348814746976767 141.71594077484252
Source 98.6342266672166 196.4367353805294 0.3559524354648185
Result 98.6342266672166 196.4367353805294 0.3559524354648185 220.7316504194533
Source 24.1328823558406 148.0674716308376 0.5553607379656484
Result 24.1328823558406 148.0674716308376 0.5553607379656484 522.3553415273689
Source 13.615131879621323 168.69545121923548 0.4703107766981316
Result 13.615131879621323 168.69545121923548 0.4703107766981316 846.2465855922111
Source 86.08613662811547 121.6463490373095 0.9964021458144853
Result 86.08613662811547 121.6463490373095 0.9964021458144853 96.02050929475305
Source 5.411924673502499 110.97323677093004 0.2638785669872925
Result 5.411924673502499 110.97323677093004 0.2638785669872925 196.85586650158865
Source 28.279099812204432 112.83945557378911 0.6787106556811437
Result 28.279099812204432 112.83945557378911 0.6787106556811437 159.4309381262589
Source 33.715164625020854 165.93172668721897 0.11975442559126814
Result 33.715164625020854 165.93172668721897 0.11975442559126814 762.230341736083
Source 68.49868039246569 108.28451299310852 0.9951405636577869
Result 68.49868039246569 108.28451299310852 0.9951405636577869 110.2121238024874
Source 78.04239880331684 199.5567274372807 0.6042143678491021
Result 78.04239880331684 199.5567274372807 0.6042143678491021 645.104564211858
Source 30.711375848172374 147.58710527981475 0.5257599360052617
Result 30.711375848172374 147.58710527981475 0.5257599360052617 267.5957752600428
Source 85.0420482262629 165.96469269907004 0.5589062761626769
Result 85.0420482262629 165.96469269907004 0.5589062761626769 611.1366218799061
Source 93.95709647571748 125.3860452946782 0.08002695540466387
Result 93.95709647571748 125.3860452946782 0.08002695540466387 118.81182444260965
Source 87.82484846140423 125.73090779395656 0.5471095051594927
Result 87.82484846140423 125.73090779395656 0.5471095051594927 83.56966990297393
Source 29.09090688375273 108.0732937108324 0.2824840592433401
Result 29.09090688375273 108.0732937108324 0.2824840592433401 183.09341893794308
Source 85.21352867987332 146.25503361900704 0.5954654927438789
Result 85.21352867987332 146.25503361900704 0.5954654927438789 142.50577461105223
Source 23.7632824835192 106.18403543347813 0.7683767493347623
Result 23.7632824835192 106.18403543347813 0.7683767493347623 149.40485867098064
Source 93.80130655523293 137.6796281210974 0.4225859081214274
Result 93.80130655523293 137.6796281210974 0.4225859081214274 225.38393256492398
Source 21.029514431942463 168.4725518501683 0.3056026448948189
Result 21.029514431942463 168.4725518501683 0.3056026448948189 459.904540081769
Source 10.709869158929674 102.87727118014759 0.5959445586823529
Result 10.709869158929674 102.87727118014759 0.5959445586823529 290.13740415663636
Source 21.978902818059044 188.6895052794776 0.4093976060198943
Result 21.978902818059044 188.6895052794776 0.4093976060198943 748.7376035114634
Source 49.93741947948669 105.02192510176728 0.3084589982327043
Result 49.93741947948669 105.02192510176728 0.3084589982327043 228.49529829087277
Выполнено 100 заданий.
simpleThreads: оба потока завершили работу
```

Рис.7

Задание 4

В 4 задании реализованы два потока Generator и Integrator, использующие общий семафор java.util.concurrent.Semaphore для синхронизации доступа к объекту Task. Generator генерирует случайные параметры задач, Integrator вычисляет интегралы. Семафор с одним разрешением обеспечивает взаимоисключающий доступ: пока один поток работает с данными, второй ожидает. Программа отрабатывает минимум 100 заданий с гарантированной синхронизацией без потерь данных. Также реализован complicatedThreads в двух режимам работы первый с прерыванием потоков через 50 миллисекунд как требуется в задании второй без прерывания для проверки полной обработки всех задач (Рис.8-11)



```
    }
    // основной метод потока
    public void run() {  new *
        Random random = new Random();// генератор случайных чисел
        int count = 0;//  счетчик для 100 задач

        try {
            while (count < 100 && !isInterrupted()) {
                try {
                    // захватываем семафор
                    semaphore.acquire();
                    if (task.getFunction() != null) {
                        semaphore.release();
                        Thread.sleep( millis: 1);
                        continue;
                    }

                    // генерация случайных параметров для задачи:
                    double a = random.nextDouble() * 9 + 1;
                    task.setFunction(new Log(a));
                    task.setLeftX(random.nextDouble() * 100);
                    task.setRightX(100 + random.nextDouble() * 100);
                    task.setStep(random.nextDouble());

                    System.out.println("Source " + task.getLeftX() + " " + task.getRightX() + " " + task.getStep());
                    count++;

                } finally {
                    // всегда освобождаем семафор
                    semaphore.release();
                }
                // пауза между генерациями
                try {
                    sleep( millis: 1);
                } catch (InterruptedException e) {
                    if (count < 100) {
                        System.out.println("Generator прерван после " + count + " задач");
                    }
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Generator error: " + e.getMessage());
        }

        System.out.println("Generator: " + count + "/100 задач");
    }
```

Рис.8

```java
public class Integrator extends Thread {  2 usages  new *
    private final Task task;  7 usages
    private final Semaphore semaphore; //семафор для синхронизации доступа  3 usages
    public Integrator(Task task, Semaphore semaphore) {  1 usage  new *
        this.task = task;
        this.semaphore = semaphore;
    }
    // основной метод потока
    public void run() {  new *
        int task_count = 0;//  счетчик обработанных задач

        try {
            // работает пока поток не прервали
            while (task_count < 100 && !isInterrupted()) {
                try {
                    // захватываем семафор
                    semaphore.acquire();

                    if (task.getFunction() != null) {
                        // чтение параметров задачи, сгенерированных Generator
                        double leftX = task.getLeftX();
                        double rightX = task.getRightX();
                        double step = task.getStep();
                        // вычисление интеграла функции на заданном интервале
                        double integral = Functions.Integral(task.getFunction(), leftX, rightX, step);
                        // вывод результата вычислений
                        System.out.println("Result " + leftX + " " + rightX + " " + step + " " + integral);

                        task.setFunction(null); // очищаем
                        task_count++;
                    }

                } finally {
                    semaphore.release();
                }
                // пауза между вычислениями
                try {
                    sleep( millis: 1);
                } catch (InterruptedException e) {
                    if (task_count < 100) {
                        System.out.println("Integrator прерван после " + task_count + " задач");
                    }
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Integrator error: " + e.getMessage());
        }

        System.out.println("Integrator: " + task_count + "/100 задач");
    }
```

Рис.9

```
Source 87.11832351973189 179.5440046158111 0.10688417840263909
Result 87.11832351973189 179.5440046158111 0.10688417840263909 583.497367269938
Source 78.82220487152291 152.46218081900324 0.7808069295371124
Result 78.82220487152291 152.46218081900324 0.7808069295371124 164.93318895931287
Source 30.814606692035017 109.2079506724856 0.13523890600094424
Result 30.814606692035017 109.2079506724856 0.13523890600094424 142.93673927672378
Source 50.15945054121879 131.0106643316024 0.07253338474893911
Result 50.15945054121879 131.0106643316024 0.07253338474893911 168.27407497322443
Source 39.098815938444844 182.29295068077192 0.6645902490652856
Result 39.098815938444844 182.29295068077192 0.6645902490652856 1366.006361189628
Source 30.153931727050676 173.55256873292814 0.6411343666118409
Result 30.153931727050676 173.55256873292814 0.6411343666118409 412.97267661799685
Source 42.90671009740432 102.89880054743065 0.022427059198961152
Result 42.90671009740432 102.89880054743065 0.022427059198961152 358.17739396922565
Source 22.425802330361456 180.27480402880673 0.33247622006050526
Result 22.425802330361456 180.27480402880673 0.33247622006050526 710.6018791622033
Source 12.367618111548884 160.3699546959349 0.4000998723088668
Result 12.367618111548884 160.3699546959349 0.4000998723088668 285.5538799300771
Source 61.9740415524115 110.32381349111998 0.3577370124786321
Result 61.9740415524115 110.32381349111998 0.3577370124786321 101.54083149141566
Source 65.5051051282283 116.54613876465352 0.08168846053985346
Result 65.5051051282283 116.54613876465352 0.08168846053985346 119.04524417603315
Source 85.0862214331265 191.32485874736213 0.9825097534216011
Result 85.0862214331265 191.32485874736213 0.9825097534216011 233.7475252649974
Source 13.515979747589002 174.36672470719367 0.750567098073011
Result 13.515979747589002 174.36672470719367 0.750567098073011 482.1987730813106
Source 92.27195415227268 137.01289790113233 0.38930629572223996
Result 92.27195415227268 137.01289790113233 0.38930629572223996 287.5892712923398
Source 11.768336504889698 187.5944905300384 0.6275738679130294
Result 11.768336504889698 187.5944905300384 0.6275738679130294 442.5133605957437
Source 85.86064854594562 145.2406306761889 0.5923685226389704
Result 85.86064854594562 145.2406306761889 0.5923685226389704 276.199923912098
Source 10.65917903463971 196.42438131468057 0.44469019150346634
Result 10.65917903463971 196.42438131468057 0.44469019150346634 3921.461407466286
Source 88.90417884169753 173.62938925734977 0.533692878087937
Result 88.90417884169753 173.62938925734977 0.533692878087937 180.7232370598612
Source 25.472306827362367 198.7429213265366 0.5081368561410087
Result 25.472306827362367 198.7429213265366 0.5081368561410087 2807.9138992298713
Source 77.02514200384681 115.12846595720065 0.739174449737623
Result 77.02514200384681 115.12846595720065 0.739174449737623 105.21844162767242
Source 69.37707733674033 191.80387225115783 0.3547928784897476
Result 69.37707733674033 191.80387225115783 0.3547928784897476 300.54150491702524
Source 79.8512597536373 120.74851838393516 0.16377702708600994
Result 79.8512597536373 120.74851838393516 0.16377702708600994 161.09370060463237
Source 18.708637686035523 144.0857836347328 0.7558432481985569
Result 18.708637686035523 144.0857836347328 0.7558432481985569 349.6304814182714
Generator: 100/100 задач
Integrator: 100/100 задач
Все 100 задач сгенерированы и проинтегрированы
```

Рис.10

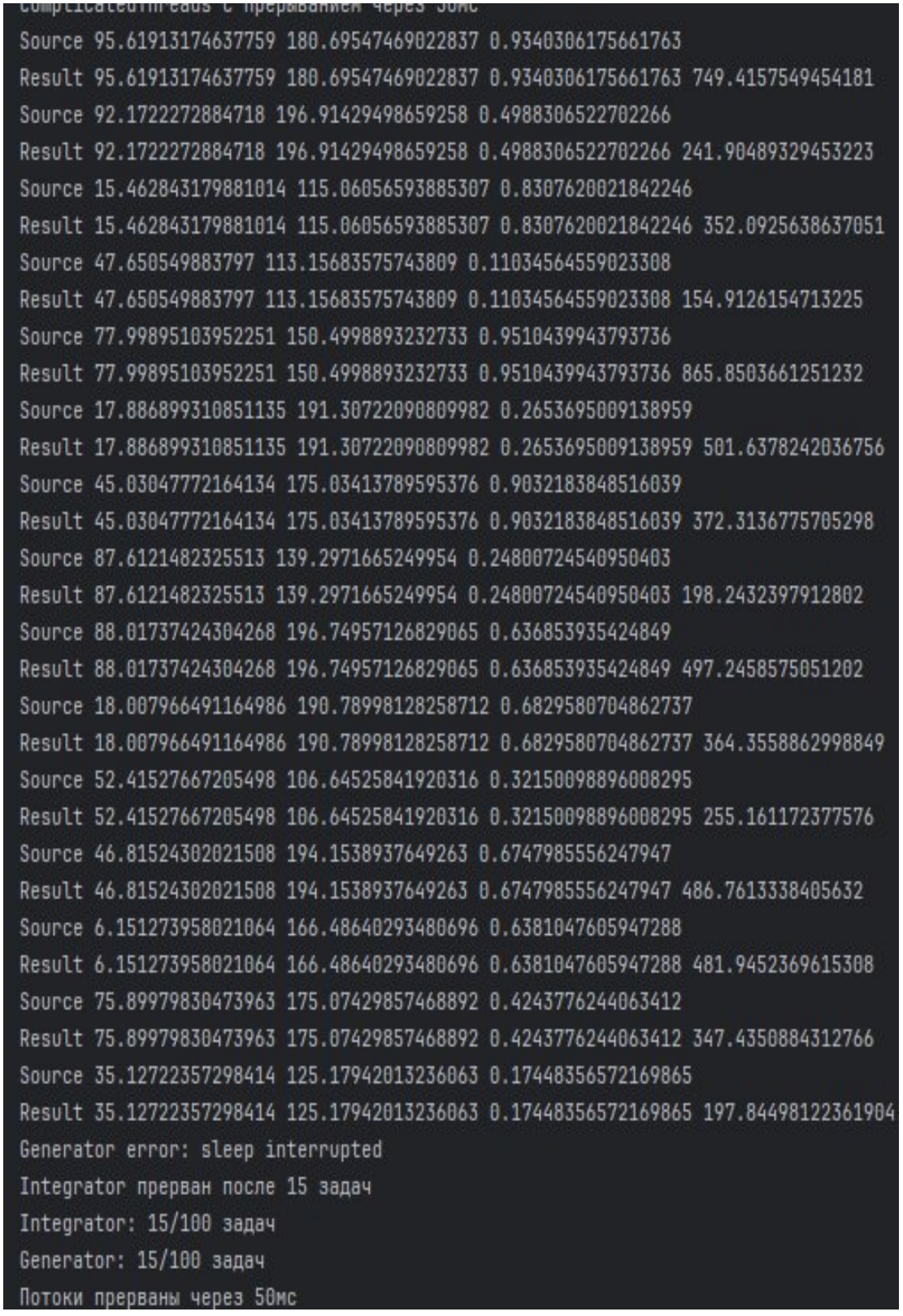

ComplicatedThreads с прерыванием через 50мс
Source 95.61913174637759 180.69547469022837 0.9340306175661763
Result 95.61913174637759 180.69547469022837 0.9340306175661763 749.4157549454181
Source 92.1722272884718 196.91429498659258 0.4988306522702266
Result 92.1722272884718 196.91429498659258 0.4988306522702266 241.90489329453223
Source 15.462843179881014 115.06056593885307 0.8307620021842246
Result 15.462843179881014 115.06056593885307 0.8307620021842246 352.09256386370 51
Source 47.650549883797 113.15683575743809 0.11034564559023308
Result 47.650549883797 113.15683575743809 0.11034564559023308 154.9126154713225
Source 77.99895103952251 150.4998893232733 0.9510439943793736
Result 77.99895103952251 150.4998893232733 0.9510439943793736 865.8503661251232
Source 17.886899310851135 191.30722090809982 0.2653695009138959
Result 17.886899310851135 191.30722090809982 0.2653695009138959 501.6378242036756
Source 45.03047772164134 175.03413789595376 0.9032183848516039
Result 45.03047772164134 175.03413789595376 0.9032183848516039 372.3136775705298
Source 87.6121482325513 139.2971665249954 0.24800724540950403
Result 87.6121482325513 139.2971665249954 0.24800724540950403 198.2432397912802
Source 88.01737424304268 196.74957126829065 0.636853935424849
Result 88.01737424304268 196.74957126829065 0.636853935424849 497.2458575051202
Source 18.007966491164986 190.78998128258712 0.6829580704862737
Result 18.007966491164986 190.78998128258712 0.6829580704862737 364.3558862998849
Source 52.41527667205498 106.64525841920316 0.32150098896008295
Result 52.41527667205498 106.64525841920316 0.32150098896008295 255.161172377576
Source 46.81524302021508 194.1538937649263 0.6747985556247947
Result 46.81524302021508 194.1538937649263 0.6747985556247947 486.7613338405632
Source 6.151273958021064 166.48640293480696 0.6381047605947288
Result 6.151273958021064 166.48640293480696 0.6381047605947288 481.9452369615308
Source 75.89979830473963 175.07429857468892 0.4243776244063412
Result 75.89979830473963 175.07429857468892 0.4243776244063412 347.4350884312766
Source 35.12722357298414 125.17942013236063 0.17448356572169865
Result 35.12722357298414 125.17942013236063 0.17448356572169865 197.84498122361904
Generator error: sleep interrupted
Integrator прерван после 15 задач
Integrator: 15/100 задач
Generator: 15/100 задач
Потоки прерваны через 50мс

Рис.11