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)

```
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;
            }
        }
    }
}
```

Ри
с.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)

```java
public class Generator extends Thread {  2 usages  new *
    private final Task task;  8 usages
    private final Semaphore semaphore; // семафор для синхронизации доступа  3 usages

    public Generator(Task task, Semaphore semaphore) {  1 usage  new *
        this.task = task;
        this.semaphore = semaphore;
    }
    // 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();

                    // 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

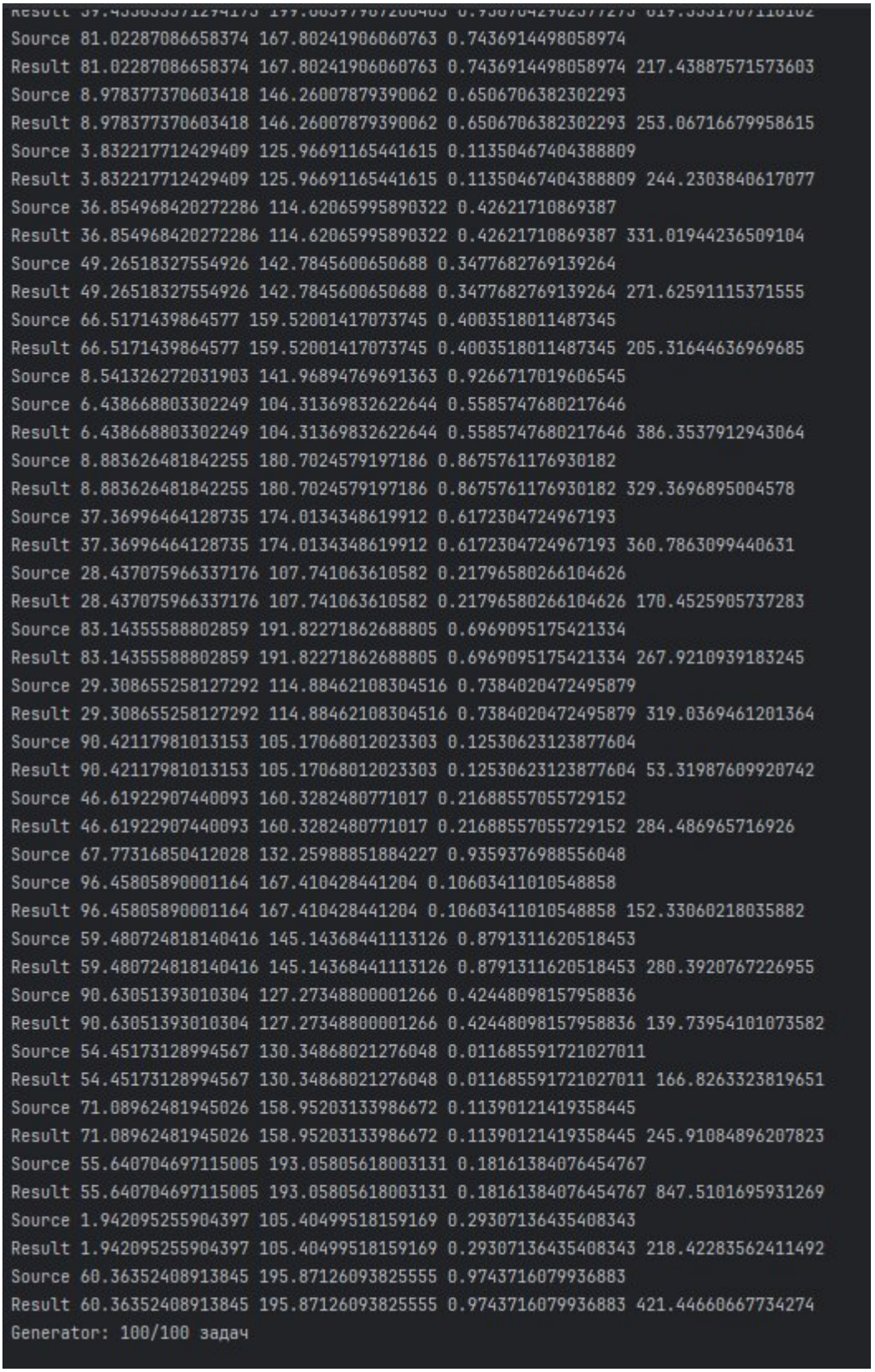
Source 81.02287086658374 167.80241906060763 0.7436914498058974
Result 81.02287086658374 167.80241906060763 0.7436914498058974 217.43887571573603
Source 8.978377370603418 146.26007879390062 0.6506706382302293
Result 8.978377370603418 146.26007879390062 0.6506706382302293 253.06716679958615
Source 3.832217712429409 125.96691165441615 0.11350467404388809
Result 3.832217712429409 125.96691165441615 0.11350467404388809 244.2303840617077
Source 36.854968420272286 114.62065995890322 0.42621710869387
Result 36.854968420272286 114.62065995890322 0.42621710869387 331.01944236509104
Source 49.26518327554926 142.7845600650688 0.3477682769139264
Result 49.26518327554926 142.7845600650688 0.3477682769139264 271.62591115371555
Source 66.5171439864577 159.52001417073745 0.4003518011487345
Result 66.5171439864577 159.52001417073745 0.4003518011487345 205.31644636969685
Source 8.541326272031903 141.96894769691363 0.9266717019606545
Source 6.438668803302249 104.31369832622644 0.5585747680217646
Result 6.438668803302249 104.31369832622644 0.5585747680217646 386.3537912943064
Source 8.883626481842255 180.7024579197186 0.8675761176930182
Result 8.883626481842255 180.7024579197186 0.8675761176930182 329.3696895004578
Source 37.36996464128735 174.0134348619912 0.6172304724967193
Result 37.36996464128735 174.0134348619912 0.6172304724967193 360.7863099440631
Source 28.437075966337176 107.741063610582 0.21796580266104626
Result 28.437075966337176 107.741063610582 0.21796580266104626 170.4525905737283
Source 83.14355588802859 191.82271862688805 0.6969095175421334
Result 83.14355588802859 191.82271862688805 0.6969095175421334 267.9210939183245
Source 29.308655258127292 114.88462108304516 0.7384020472495879
Result 29.308655258127292 114.88462108304516 0.7384020472495879 319.0369461201364
Source 90.42117981013153 105.17068012023303 0.12530623123877604
Result 90.42117981013153 105.17068012023303 0.12530623123877604 53.31987609920742
Source 46.61922907440093 160.3282480771017 0.21688557055729152
Result 46.61922907440093 160.3282480771017 0.21688557055729152 284.486965716926
Source 67.77316850412028 132.25988851884227 0.9359376988556048
Source 96.45805890001164 167.410428441204 0.10603411010548858
Result 96.45805890001164 167.410428441204 0.10603411010548858 152.33060218035882
Source 59.480724818140416 145.14368441113126 0.8791311620518453
Result 59.480724818140416 145.14368441113126 0.8791311620518453 280.3920767226955
Source 90.63051393010304 127.27348800001266 0.42448098157958836
Result 90.63051393010304 127.27348800001266 0.42448098157958836 139.73954101073582
Source 54.45173128994567 130.34868021276048 0.011685591721027011
Result 54.45173128994567 130.34868021276048 0.011685591721027011 166.8263323819651
Source 71.08962481945026 158.95203133986672 0.11390121419358445
Result 71.08962481945026 158.95203133986672 0.11390121419358445 245.91084896207823
Source 55.640704697115005 193.05805618003131 0.18161384076454767
Result 55.640704697115005 193.05805618003131 0.18161384076454767 847.5101695931269
Source 1.942095255904397 105.40499518159169 0.29307136435408343
Result 1.942095255904397 105.40499518159169 0.29307136435408343 218.42283562411492
Source 60.36352408913845 195.87126093825555 0.9743716079936883
Result 60.36352408913845 195.87126093825555 0.9743716079936883 421.44660667734274
Generator: 100/100 задач

Рис.10

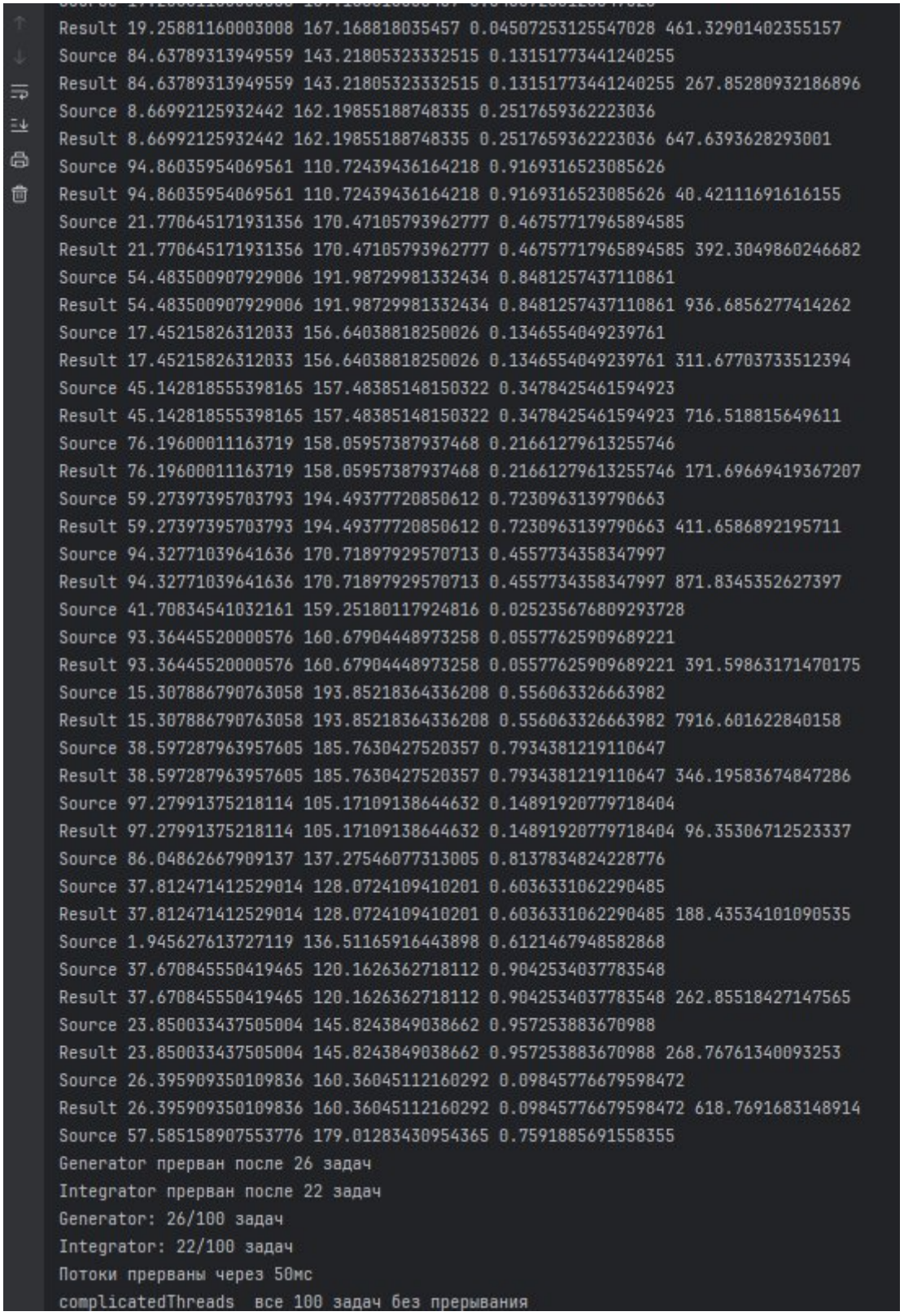
Result 19.25881160003008 167.168818035457 0.04507253125547028 461.32901402355157
Source 84.63789313949559 143.21805323332515 0.13151773441240255
Result 84.63789313949559 143.21805323332515 0.13151773441240255 267.85280932186896
Source 8.66992125932442 162.19855188748335 0.2517659362223036
Result 8.66992125932442 162.19855188748335 0.2517659362223036 647.6393628293001
Source 94.86035954069561 110.72439436164218 0.9169316523085626
Result 94.86035954069561 110.72439436164218 0.9169316523085626 40.42111691616155
Source 21.770645171931356 170.47105793962777 0.46757717965894585
Result 21.770645171931356 170.47105793962777 0.46757717965894585 392.3049860246682
Source 54.483500907929006 191.98729981332434 0.8481257437110861
Result 54.483500907929006 191.98729981332434 0.8481257437110861 936.6856277414262
Source 17.45215826312033 156.64038818250026 0.1346554049239761
Result 17.45215826312033 156.64038818250026 0.1346554049239761 311.67703733512394
Source 45.142818555398165 157.48385148150322 0.3478425461594923
Result 45.142818555398165 157.48385148150322 0.3478425461594923 716.518815649611
Source 76.19600011163719 158.05957387937468 0.21661279613255746
Result 76.19600011163719 158.05957387937468 0.21661279613255746 171.69669419367207
Source 59.27397395703793 194.49377720850612 0.7230963139790663
Result 59.27397395703793 194.49377720850612 0.7230963139790663 411.6586892195711
Source 94.32771039641636 170.71897929570713 0.4557734358347997
Result 94.32771039641636 170.71897929570713 0.4557734358347997 871.8345352627397
Source 41.70834541032161 159.25180117924816 0.025235676809293728
Source 93.36445520000576 160.67904448973258 0.05577625909689221
Result 93.36445520000576 160.67904448973258 0.05577625909689221 391.59863171470175
Source 15.307886790763058 193.85218364336208 0.556063326663982
Result 15.307886790763058 193.85218364336208 0.556063326663982 7916.601622840158
Source 38.597287963957605 185.7630427520357 0.7934381219110647
Result 38.597287963957605 185.7630427520357 0.7934381219110647 346.19583674847286
Source 97.27991375218114 105.17109138644632 0.14891920779718404
Result 97.27991375218114 105.17109138644632 0.14891920779718404 96.35306712523337
Source 86.04862667909137 137.27546077313005 0.8137834824228776
Source 37.812471412529014 128.0724109410201 0.6036331062290485
Result 37.812471412529014 128.0724109410201 0.6036331062290485 188.43534101090535
Source 1.945627613727119 136.51165916443898 0.6121467948582868
Source 37.670845550419465 120.1626362718112 0.9042534037783548
Result 37.670845550419465 120.1626362718112 0.9042534037783548 262.85518427147565
Source 23.850033437505004 145.8243849038662 0.957253883670988
Result 23.850033437505004 145.8243849038662 0.957253883670988 268.76761340093253
Source 26.395909350109836 160.36045112160292 0.09845776679598472
Result 26.395909350109836 160.36045112160292 0.09845776679598472 618.7691683148914
Source 57.585158907553776 179.01283430954365 0.7591885691558355
Generator прерван после 26 задач
Integrator прерван после 22 задач
Generator: 26/100 задач
Integrator: 22/100 задач
Потоки прерваны через 50мс
complicatedThreads все 100 задач без прерывания

Рис.11