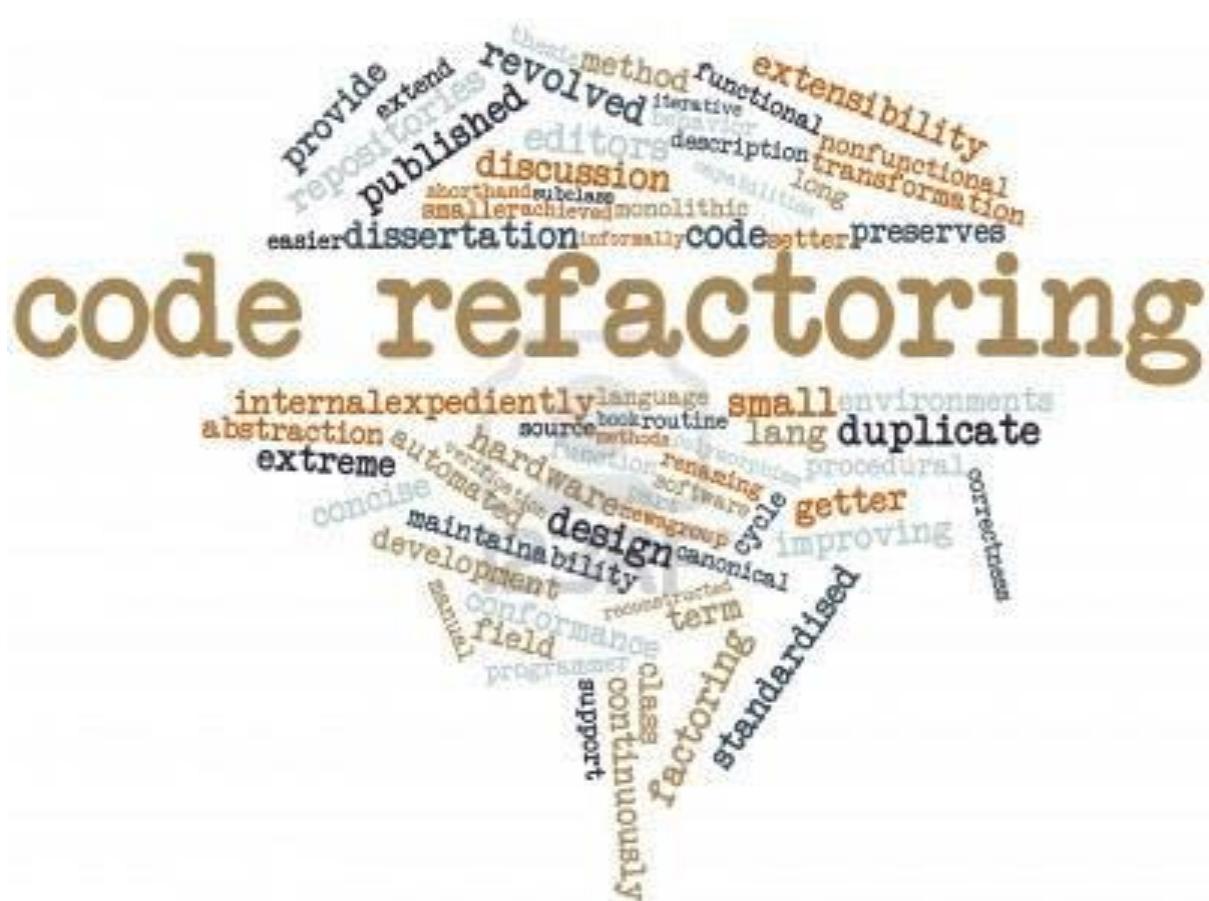


פרויקט מסכם

מבוא לפירוק והרכבת תוכניות



מגישה: אור-אל שחר

הסבר כללי על הפרויקט:

בפרויקט השתמשתי בפרויקט קוד פתוח - FreeChart 1.0.10 אשר לקחת מהאתר של הפרויקט [Clone Refactoring with Lambda Expressions](#) ניתן למצוא את הקוד שלו גם [Github](#).

הפרויקט הוא בשפת java.

בפרויקט הדגמתי 4 דוגמאות שונות.

בפרויקט אדגים את השימוש את שיטות הריפקטור:

- Split loop
- Replace Temp with query
- Type 3 – Clone Elimination
- inline method
- inline temp
- Extract method

ביצענו זאת ע"י שימוש באלגוריתמים שלמדנו בכיתה תוך מתן הסבר לביצוע כל שלב
וכיצד ביצענו זאת.

השתמשנו גם ב Sliding Bucketing וגם ב Bucketing לאורך כל הפרויקט ניסיתי להראות כמה
שיטות ולהציג מהי חזקתו ומהי חולשתו של כל אלגוריתם לעומת השני וזאת ביצעת
ע' הפעלת האלגוריתם על אותו הקוד כדי לראות ולזהות באיזה נקודות כל אלגוריתם
חזק ובאיזה חלש.

פיתחנו דיוון לאחר כל דוגמה – הסבכנו האם השימוש באלגוריתם תרם, האם השתלים
ומה היתרונות והחסרונות עבור הקוד שקיבלנו.

תיעוד השלבים –

לאורך הפרויקט נוכל לראות כיצד ביצענו את כל אחד משלבי האלגוריתם (האלגוריתמים
מצורפים בדוגמה הראשונה של כל קוד), את הגрафים השונים ואת הפלט של
האלגוריתמים.

טסטיים-

חלק נכבד לפני כל ביצוע אלגוריתם היה ביצוע של סדרת טסטיים בה ניסינו להכשיל במכoon את הטסטיים הקיימים על מנת לבדוק שלאחר כל שינוי שביצענו קימפלנו את הקוד ובכך שהטסטיים עדין עוברים.

בפרויקט הנ"ל התמקדתי במחלקות הבאות:

//jfreechart-1.0.10/src/org/jfree/data/statistics/Regression.java/

/jfreechart-1.0.10/src/org/jfree/chart/renderer/xy/XYErrorRenderer.java/

מחלקות הטסטיים אותן הרצתי לאורך הפרויקט הן:

/jfreechart-1.0.10/tests/org/jfree/data/statistics/junit/RegressionTests.java/

/jfreechart- /

1.0.10/tests/org/jfree/chart/renderer/xy/junit/XYErrorRendererTests.java

Split loop – direct application of sliding

מחלקות קוד: Regression.java

מחלקות טסטים: RegressionTests.java

מטרת הפונקציה:

מחזירה מערך בגודל 2 שמכיל 2 ערכים:

למשווהה של הרגסיה כלומר מחזירה את ערכי a ו-b עבור המשוואה $y = ax^b$

כasher [0]=a , result[1]=b result

מטרת הריבקטור הנ"ל:

loop Split – מציג באופן מובהק את השימוש של אלגוריתם sliding נוכל לראות שבולולאה יש לנו חישוב של 4 ערכים שונים.

נרצה לפצל את החישובים של כל אחד ללואה נפרדת תוך שמירה על זרימת המידע וששומ דבר לא הרט. (נבחן כי אין תלות מידע ביניהם.)

The screenshot shows a Java code editor with the file `Regression.java` open. The code implements a power regression algorithm to find parameters a and b for the equation $y = ax^b$. The code uses logarithmic transformation and calculates sums of powers and products of data points. It handles cases where there are fewer than 2 data points and returns the result as a double array [a, b]. The code is annotated with Javadoc-style comments and annotations.

```
183     /**
184      * Returns the parameters 'a' and 'b' for an equation  $y = ax^b$ , fitted to
185      * the data using a power regression equation. The result is returned as
186      * an array, where  $double[0] \rightarrow a$ , and  $double[1] \rightarrow b$ .
187      *
188      * @param data the data.
189      * @param series the series to fit the regression line against.
190      *
191      * @return The parameters.
192      */
193     public static double[] getPowerRegression(XYDataset data, int series) {
194
195         int n = data.getItemCount(series);
196         if (n < 2) {
197             throw new IllegalArgumentException("Not enough data.");
198         }
199
200         double sumX = 0;
201         double sumY = 0;
202         double sumXX = 0;
203         double sumXY = 0;
204         for (int i = 0; i < n; i++) {
205             double x = Math.log(data.getValue(series, i));
206             double y = Math.log(data.getYValue(series, i));
207             sumX += x;
208             sumY += y;
209             double xx = x * x;
210             sumXX += xx;
211             double xy = x * y;
212             sumXY += xy;
213         }
214         double sxx = sumXX - (sumX * sumX) / n;
215         double sxy = sumXY - (sumX * sumY) / n;
216         double xbar = sumX / n;
217         double ybar = sumY / n;
218
219         double[] result = new double[2];
220         result[1] = sxy / sxx;
221         result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);
222
223         return result;
224     }
225 }
226
227 }
228 }
```

Regression → getPowerRegression()

טסיטים ראשוניים:

תחליה נבדוק שאכן יש טסיטים שמכסים את הפונקציה הנ"ל ושבמידה ונחרוס שהוא במהלך ביצוע הריפקטור אכן בעזרת הטסיטים נוכל לגלוות טעות זו.

- ביצענו מספר שינויים בקוד: 6 סה"כ
 - 4 מהם שינויו את סדר זרימת המידע בין המשתנים השונים אותם נרצה לפרש כלולאה.
 - 2 שינויים בערך של x ו-y כדי לבדוק שאכן הטסיטים נופלים (כיוון שהערכים של המשתנים אכן תלויים בערכם של x ו-y).
 - להן תמונות של הפונקציה עם השינויים השונים שביצענו עליהם וכיידם נופלים או עוברים לפי מה שבדקנו:

The screenshot shows an IDE interface with two main panes. The left pane displays the `Regression.java` file, which contains a static method `getPowerRegression` that calculates a power regression. The right pane displays the `RegressionTests.java` file, which contains several test methods for regression calculations. Below these panes is a "Run" window showing the execution results: 2 tests failed out of 8, with a total duration of 27 ms. The failed tests are listed under "Test Results". The bottom status bar indicates the process finished with an exit code of 255.

```
Regression.java
185     /*stage 0 - initiate errors for tests
186     try 1 - replace order sumY before sumX - work ok as should
187     try 2 - replace order sumXY before sumX- work ok as should
188     try 3 - replace order sumXX before sumX - work ok as should
189     try 4 - replace order sumXY before sumY
190     try 5 - change value of x
191     try 6 - change value of y
192     */
193     public static double[] getPowerRegression(double[][] data) {
194
195         int n = data.length;
196         if (n < 2) {
197             throw new IllegalArgumentException("Not enough data.");
198         }
199
200         double sumX = 0;
201         double sumY = 0;
202         double sumXX = 0;
203         double sumXY = 0;
204         for (int i = 0; i < n; i++) {
205             //double x = Math.log(data[i][0]);
206             double x = 1;
207
208             double y = Math.log(data[i][1]);
209
210             sumX += x;
211
212             sumY += y;
213             double xx = x * x;
214             sumXX += xx;
215             double xy = x * y;
216             sumXY += xy;
217         }
218     }
219
220     public static void main(String[] args) {
221
222     }
```

```
RegressionTests.java
89     /**
90      * Checks the results of an OLS regression on sample dataset 1 AFTER
91      * converting it to an XYSeries.
92      */
93     public void testOLSRegression1a() {
94
95         double[][] data = createSampleData1();
96         double[] result = Regression.getPowerRegression(data);
97         assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
98         assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
99     }
100
101    /**
102     * Checks the results of a power regression on sample dataset 1,
103     * after converting it to an XYSeries.
104     */
105    public void testPowerRegression1a() {
106
107        double[][] data = createSampleData1();
108        XYSeries series = new XYSeries( key: "Test" );
109        for (int i = 0; i < 11; i++) {
110            series.add(data[i][0], data[i][1]);
111        }
112        XYDataset ds = new XYDatasetCollection(series);
113        double[] result = Regression.getPowerRegression(ds, series: 0);
114
115        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
116        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
117    }
118
119    /**
120     * Checks the results of a power regression on sample dataset 1 AFTER
121     * converting it to an XYSeries.
122     */
123    public void testPowerRegression1b() {
124
125        double[][] data = createSampleData1();
126
127        XYSeries series = new XYSeries( key: "Test" );
128        for (int i = 0; i < 11; i++) {
129            series.add(data[i][0], data[i][1]);
130        }
131        XYDataset ds = new XYDatasetCollection(series);
132        double[] result = Regression.getPowerRegression(ds, series: 1);
133
134        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
135        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
136    }
137
138    /**
139     * Checks the results of an OLS regression on sample dataset 1 AFTER
140     * converting it to an XYSeries.
141     */
142    public void testOLSRegression1b() {
143
144        double[][] data = createSampleData1();
145        XYSeries series = new XYSeries( key: "Test" );
146        for (int i = 0; i < 11; i++) {
147            series.add(data[i][0], data[i][1]);
148        }
149        XYDataset ds = new XYDatasetCollection(series);
150        double[] result = Regression.getPowerRegression(ds, series: 1);
151
152        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
153        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
154    }
155
156    /**
157     * Checks the results of a power regression on sample dataset 1 AFTER
158     * converting it to an XYSeries.
159     */
160    public void testPowerRegression1b() {
161
162        double[][] data = createSampleData1();
163
164        XYSeries series = new XYSeries( key: "Test" );
165        for (int i = 0; i < 11; i++) {
166            series.add(data[i][0], data[i][1]);
167        }
168        XYDataset ds = new XYDatasetCollection(series);
169        double[] result = Regression.getPowerRegression(ds, series: 1);
170
171        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
172        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
173    }
174
175    /**
176     * Checks the results of an OLS regression on sample dataset 1 AFTER
177     * converting it to an XYSeries.
178     */
179    public void testOLSRegression2a() {
180
181        double[][] data = createSampleData1();
182
183        XYSeries series = new XYSeries( key: "Test" );
184        for (int i = 0; i < 11; i++) {
185            series.add(data[i][0], data[i][1]);
186        }
187        XYDataset ds = new XYDatasetCollection(series);
188        double[] result = Regression.getPowerRegression(ds, series: 0);
189
190        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
191        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
192    }
193
194    /**
195     * Checks the results of a power regression on sample dataset 1 AFTER
196     * converting it to an XYSeries.
197     */
198    public void testPowerRegression2a() {
199
200        double[][] data = createSampleData1();
201
202        XYSeries series = new XYSeries( key: "Test" );
203        for (int i = 0; i < 11; i++) {
204            series.add(data[i][0], data[i][1]);
205        }
206        XYDataset ds = new XYDatasetCollection(series);
207        double[] result = Regression.getPowerRegression(ds, series: 1);
208
209        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
210        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
211    }
212
213    /**
214     * Checks the results of an OLS regression on sample dataset 1 AFTER
215     * converting it to an XYSeries.
216     */
217    public void testOLSRegression2b() {
218
219        double[][] data = createSampleData1();
220
221        XYSeries series = new XYSeries( key: "Test" );
222        for (int i = 0; i < 11; i++) {
223            series.add(data[i][0], data[i][1]);
224        }
225        XYDataset ds = new XYDatasetCollection(series);
226        double[] result = Regression.getPowerRegression(ds, series: 0);
227
228        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
229        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
230    }
231
232    /**
233     * Checks the results of a power regression on sample dataset 1 AFTER
234     * converting it to an XYSeries.
235     */
236    public void testPowerRegression2b() {
237
238        double[][] data = createSampleData1();
239
240        XYSeries series = new XYSeries( key: "Test" );
241        for (int i = 0; i < 11; i++) {
242            series.add(data[i][0], data[i][1]);
243        }
244        XYDataset ds = new XYDatasetCollection(series);
245        double[] result = Regression.getPowerRegression(ds, series: 1);
246
247        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
248        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
249    }
250
251    /**
252     * Checks the results of an OLS regression on sample dataset 1 AFTER
253     * converting it to an XYSeries.
254     */
255    public void testOLSRegression3a() {
256
257        double[][] data = createSampleData1();
258
259        XYSeries series = new XYSeries( key: "Test" );
260        for (int i = 0; i < 11; i++) {
261            series.add(data[i][0], data[i][1]);
262        }
263        XYDataset ds = new XYDatasetCollection(series);
264        double[] result = Regression.getPowerRegression(ds, series: 0);
265
266        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
267        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
268    }
269
270    /**
271     * Checks the results of a power regression on sample dataset 1 AFTER
272     * converting it to an XYSeries.
273     */
274    public void testPowerRegression3a() {
275
276        double[][] data = createSampleData1();
277
278        XYSeries series = new XYSeries( key: "Test" );
279        for (int i = 0; i < 11; i++) {
280            series.add(data[i][0], data[i][1]);
281        }
282        XYDataset ds = new XYDatasetCollection(series);
283        double[] result = Regression.getPowerRegression(ds, series: 1);
284
285        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
286        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
287    }
288
289    /**
290     * Checks the results of an OLS regression on sample dataset 1 AFTER
291     * converting it to an XYSeries.
292     */
293    public void testOLSRegression3b() {
294
295        double[][] data = createSampleData1();
296
297        XYSeries series = new XYSeries( key: "Test" );
298        for (int i = 0; i < 11; i++) {
299            series.add(data[i][0], data[i][1]);
300        }
301        XYDataset ds = new XYDatasetCollection(series);
302        double[] result = Regression.getPowerRegression(ds, series: 0);
303
304        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
305        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
306    }
307
308    /**
309     * Checks the results of a power regression on sample dataset 1 AFTER
310     * converting it to an XYSeries.
311     */
312    public void testPowerRegression3b() {
313
314        double[][] data = createSampleData1();
315
316        XYSeries series = new XYSeries( key: "Test" );
317        for (int i = 0; i < 11; i++) {
318            series.add(data[i][0], data[i][1]);
319        }
320        XYDataset ds = new XYDatasetCollection(series);
321        double[] result = Regression.getPowerRegression(ds, series: 1);
322
323        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
324        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
325    }
326
327    /**
328     * Checks the results of an OLS regression on sample dataset 1 AFTER
329     * converting it to an XYSeries.
330     */
331    public void testOLSRegression4a() {
332
333        double[][] data = createSampleData1();
334
335        XYSeries series = new XYSeries( key: "Test" );
336        for (int i = 0; i < 11; i++) {
337            series.add(data[i][0], data[i][1]);
338        }
339        XYDataset ds = new XYDatasetCollection(series);
340        double[] result = Regression.getPowerRegression(ds, series: 0);
341
342        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
343        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
344    }
345
346    /**
347     * Checks the results of a power regression on sample dataset 1 AFTER
348     * converting it to an XYSeries.
349     */
350    public void testPowerRegression4a() {
351
352        double[][] data = createSampleData1();
353
354        XYSeries series = new XYSeries( key: "Test" );
355        for (int i = 0; i < 11; i++) {
356            series.add(data[i][0], data[i][1]);
357        }
358        XYDataset ds = new XYDatasetCollection(series);
359        double[] result = Regression.getPowerRegression(ds, series: 1);
360
361        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
362        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
363    }
364
365    /**
366     * Checks the results of an OLS regression on sample dataset 1 AFTER
367     * converting it to an XYSeries.
368     */
369    public void testOLSRegression4b() {
370
371        double[][] data = createSampleData1();
372
373        XYSeries series = new XYSeries( key: "Test" );
374        for (int i = 0; i < 11; i++) {
375            series.add(data[i][0], data[i][1]);
376        }
377        XYDataset ds = new XYDatasetCollection(series);
378        double[] result = Regression.getPowerRegression(ds, series: 0);
379
380        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
381        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
382    }
383
384    /**
385     * Checks the results of a power regression on sample dataset 1 AFTER
386     * converting it to an XYSeries.
387     */
388    public void testPowerRegression4b() {
389
390        double[][] data = createSampleData1();
391
392        XYSeries series = new XYSeries( key: "Test" );
393        for (int i = 0; i < 11; i++) {
394            series.add(data[i][0], data[i][1]);
395        }
396        XYDataset ds = new XYDatasetCollection(series);
397        double[] result = Regression.getPowerRegression(ds, series: 1);
398
399        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
400        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
401    }
402
403    /**
404     * Checks the results of an OLS regression on sample dataset 1 AFTER
405     * converting it to an XYSeries.
406     */
407    public void testOLSRegression5a() {
408
409        double[][] data = createSampleData1();
410
411        XYSeries series = new XYSeries( key: "Test" );
412        for (int i = 0; i < 11; i++) {
413            series.add(data[i][0], data[i][1]);
414        }
415        XYDataset ds = new XYDatasetCollection(series);
416        double[] result = Regression.getPowerRegression(ds, series: 0);
417
418        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
419        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
420    }
421
422    /**
423     * Checks the results of a power regression on sample dataset 1 AFTER
424     * converting it to an XYSeries.
425     */
426    public void testPowerRegression5a() {
427
428        double[][] data = createSampleData1();
429
430        XYSeries series = new XYSeries( key: "Test" );
431        for (int i = 0; i < 11; i++) {
432            series.add(data[i][0], data[i][1]);
433        }
434        XYDataset ds = new XYDatasetCollection(series);
435        double[] result = Regression.getPowerRegression(ds, series: 1);
436
437        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
438        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
439    }
440
441    /**
442     * Checks the results of an OLS regression on sample dataset 1 AFTER
443     * converting it to an XYSeries.
444     */
445    public void testOLSRegression5b() {
446
447        double[][] data = createSampleData1();
448
449        XYSeries series = new XYSeries( key: "Test" );
450        for (int i = 0; i < 11; i++) {
451            series.add(data[i][0], data[i][1]);
452        }
453        XYDataset ds = new XYDatasetCollection(series);
454        double[] result = Regression.getPowerRegression(ds, series: 0);
455
456        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
457        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
458    }
459
460    /**
461     * Checks the results of a power regression on sample dataset 1 AFTER
462     * converting it to an XYSeries.
463     */
464    public void testPowerRegression5b() {
465
466        double[][] data = createSampleData1();
467
468        XYSeries series = new XYSeries( key: "Test" );
469        for (int i = 0; i < 11; i++) {
470            series.add(data[i][0], data[i][1]);
471        }
472        XYDataset ds = new XYDatasetCollection(series);
473        double[] result = Regression.getPowerRegression(ds, series: 1);
474
475        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
476        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
477    }
478
479    /**
480     * Checks the results of an OLS regression on sample dataset 1 AFTER
481     * converting it to an XYSeries.
482     */
483    public void testOLSRegression6a() {
484
485        double[][] data = createSampleData1();
486
487        XYSeries series = new XYSeries( key: "Test" );
488        for (int i = 0; i < 11; i++) {
489            series.add(data[i][0], data[i][1]);
490        }
491        XYDataset ds = new XYDatasetCollection(series);
492        double[] result = Regression.getPowerRegression(ds, series: 0);
493
494        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
495        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
496    }
497
498    /**
499     * Checks the results of a power regression on sample dataset 1 AFTER
500     * converting it to an XYSeries.
501     */
502    public void testPowerRegression6a() {
503
504        double[][] data = createSampleData1();
505
506        XYSeries series = new XYSeries( key: "Test" );
507        for (int i = 0; i < 11; i++) {
508            series.add(data[i][0], data[i][1]);
509        }
510        XYDataset ds = new XYDatasetCollection(series);
511        double[] result = Regression.getPowerRegression(ds, series: 1);
512
513        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
514        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
515    }
516
517    /**
518     * Checks the results of an OLS regression on sample dataset 1 AFTER
519     * converting it to an XYSeries.
520     */
521    public void testOLSRegression6b() {
522
523        double[][] data = createSampleData1();
524
525        XYSeries series = new XYSeries( key: "Test" );
526        for (int i = 0; i < 11; i++) {
527            series.add(data[i][0], data[i][1]);
528        }
529        XYDataset ds = new XYDatasetCollection(series);
530        double[] result = Regression.getPowerRegression(ds, series: 0);
531
532        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
533        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
534    }
535
536    /**
537     * Checks the results of a power regression on sample dataset 1 AFTER
538     * converting it to an XYSeries.
539     */
540    public void testPowerRegression6b() {
541
542        double[][] data = createSampleData1();
543
544        XYSeries series = new XYSeries( key: "Test" );
545        for (int i = 0; i < 11; i++) {
546            series.add(data[i][0], data[i][1]);
547        }
548        XYDataset ds = new XYDatasetCollection(series);
549        double[] result = Regression.getPowerRegression(ds, series: 1);
550
551        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
552        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
553    }
554
555    /**
556     * Checks the results of an OLS regression on sample dataset 1 AFTER
557     * converting it to an XYSeries.
558     */
559    public void testOLSRegression7a() {
560
561        double[][] data = createSampleData1();
562
563        XYSeries series = new XYSeries( key: "Test" );
564        for (int i = 0; i < 11; i++) {
565            series.add(data[i][0], data[i][1]);
566        }
567        XYDataset ds = new XYDatasetCollection(series);
568        double[] result = Regression.getPowerRegression(ds, series: 0);
569
570        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
571        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
572    }
573
574    /**
575     * Checks the results of a power regression on sample dataset 1 AFTER
576     * converting it to an XYSeries.
577     */
578    public void testPowerRegression7a() {
579
580        double[][] data = createSampleData1();
581
582        XYSeries series = new XYSeries( key: "Test" );
583        for (int i = 0; i < 11; i++) {
584            series.add(data[i][0], data[i][1]);
585        }
586        XYDataset ds = new XYDatasetCollection(series);
587        double[] result = Regression.getPowerRegression(ds, series: 1);
588
589        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
590        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
591    }
592
593    /**
594     * Checks the results of an OLS regression on sample dataset 1 AFTER
595     * converting it to an XYSeries.
596     */
597    public void testOLSRegression7b() {
598
599        double[][] data = createSampleData1();
600
601        XYSeries series = new XYSeries( key: "Test" );
602        for (int i = 0; i < 11; i++) {
603            series.add(data[i][0], data[i][1]);
604        }
605        XYDataset ds = new XYDatasetCollection(series);
606        double[] result = Regression.getPowerRegression(ds, series: 0);
607
608        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
609        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
610    }
611
612    /**
613     * Checks the results of a power regression on sample dataset 1 AFTER
614     * converting it to an XYSeries.
615     */
616    public void testPowerRegression7b() {
617
618        double[][] data = createSampleData1();
619
620        XYSeries series = new XYSeries( key: "Test" );
621        for (int i = 0; i < 11; i++) {
622            series.add(data[i][0], data[i][1]);
623        }
624        XYDataset ds = new XYDatasetCollection(series);
625        double[] result = Regression.getPowerRegression(ds, series: 1);
626
627        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
628        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
629    }
630
631    /**
632     * Checks the results of an OLS regression on sample dataset 1 AFTER
633     * converting it to an XYSeries.
634     */
635    public void testOLSRegression8a() {
636
637        double[][] data = createSampleData1();
638
639        XYSeries series = new XYSeries( key: "Test" );
640        for (int i = 0; i < 11; i++) {
641            series.add(data[i][0], data[i][1]);
642        }
643        XYDataset ds = new XYDatasetCollection(series);
644        double[] result = Regression.getPowerRegression(ds, series: 0);
645
646        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
647        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
648    }
649
650    /**
651     * Checks the results of a power regression on sample dataset 1 AFTER
652     * converting it to an XYSeries.
653     */
654    public void testPowerRegression8a() {
655
656        double[][] data = createSampleData1();
657
658        XYSeries series = new XYSeries( key: "Test" );
659        for (int i = 0; i < 11; i++) {
660            series.add(data[i][0], data[i][1]);
661        }
662        XYDataset ds = new XYDatasetCollection(series);
663        double[] result = Regression.getPowerRegression(ds, series: 1);
664
665        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
666        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
667    }
668
669    /**
670     * Checks the results of an OLS regression on sample dataset 1 AFTER
671     * converting it to an XYSeries.
672     */
673    public void testOLSRegression8b() {
674
675        double[][] data = createSampleData1();
676
677        XYSeries series = new XYSeries( key: "Test" );
678        for (int i = 0; i < 11; i++) {
679            series.add(data[i][0], data[i][1]);
680        }
681        XYDataset ds = new XYDatasetCollection(series);
682        double[] result = Regression.getPowerRegression(ds, series: 0);
683
684        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
685        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
686    }
687
688    /**
689     * Checks the results of a power regression on sample dataset 1 AFTER
690     * converting it to an XYSeries.
691     */
692    public void testPowerRegression8b() {
693
694        double[][] data = createSampleData1();
695
696        XYSeries series = new XYSeries( key: "Test" );
697        for (int i = 0; i < 11; i++) {
698            series.add(data[i][0], data[i][1]);
699        }
700        XYDataset ds = new XYDatasetCollection(series);
701        double[] result = Regression.getPowerRegression(ds, series: 1);
702
703        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
704        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
705    }
706
707    /**
708     * Checks the results of an OLS regression on sample dataset 1 AFTER
709     * converting it to an XYSeries.
710     */
711    public void testOLSRegression9a() {
712
713        double[][] data = createSampleData1();
714
715        XYSeries series = new XYSeries( key: "Test" );
716        for (int i = 0; i < 11; i++) {
717            series.add(data[i][0], data[i][1]);
718        }
719        XYDataset ds = new XYDatasetCollection(series);
720        double[] result = Regression.getPowerRegression(ds, series: 0);
721
722        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
723        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
724    }
725
726    /**
727     * Checks the results of a power regression on sample dataset 1 AFTER
728     * converting it to an XYSeries.
729     */
730    public void testPowerRegression9a() {
731
732        double[][] data = createSampleData1();
733
734        XYSeries series = new XYSeries( key: "Test" );
735        for (int i = 0; i < 11; i++) {
736            series.add(data[i][0], data[i][1]);
737        }
738        XYDataset ds = new XYDatasetCollection(series);
739        double[] result = Regression.getPowerRegression(ds, series: 1);
740
741        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
742        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
743    }
744
745    /**
746     * Checks the results of an OLS regression on sample dataset 1 AFTER
747     * converting it to an XYSeries.
748     */
749    public void testOLSRegression9b() {
750
751        double[][] data = createSampleData1();
752
753        XYSeries series = new XYSeries( key: "Test" );
754        for (int i = 0; i < 11; i++) {
755            series.add(data[i][0], data[i][1]);
756        }
757        XYDataset ds = new XYDatasetCollection(series);
758        double[] result = Regression.getPowerRegression(ds, series: 0);
759
760        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
761        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
762    }
763
764    /**
765     * Checks the results of a power regression on sample dataset 1 AFTER
766     * converting it to an XYSeries.
767     */
768    public void testPowerRegression9b() {
769
770        double[][] data = createSampleData1();
771
772        XYSeries series = new XYSeries( key: "Test" );
773        for (int i = 0; i < 11; i++) {
774            series.add(data[i][0], data[i][1]);
775        }
776        XYDataset ds = new XYDatasetCollection(series);
777        double[] result = Regression.getPowerRegression(ds, series: 1);
778
779        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
780        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
781    }
782
783    /**
784     * Checks the results of an OLS regression on sample dataset 1 AFTER
785     * converting it to an XYSeries.
786     */
787    public void testOLSRegression10a() {
788
789        double[][] data = createSampleData1();
790
791        XYSeries series = new XYSeries( key: "Test" );
792        for (int i = 0; i < 11; i++) {
793            series.add(data[i][0], data[i][1]);
794        }
795        XYDataset ds = new XYDatasetCollection(series);
796        double[] result = Regression.getPowerRegression(ds, series: 0);
797
798        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
799        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
800    }
801
802    /**
803     * Checks the results of a power regression on sample dataset 1 AFTER
804     * converting it to an XYSeries.
805     */
806    public void testPowerRegression10a() {
807
808        double[][] data = createSampleData1();
809
810        XYSeries series = new XYSeries( key: "Test" );
811        for (int i = 0; i < 11; i++) {
812            series.add(data[i][0], data[i][1]);
813        }
814        XYDataset ds = new XYDatasetCollection(series);
815        double[] result = Regression.getPowerRegression(ds, series: 1);
816
817        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
818        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
819    }
820
821    /**
822     * Checks the results of an OLS regression on sample dataset 1 AFTER
823     * converting it to an XYSeries.
824     */
825    public void testOLSRegression10b() {
826
827        double[][] data = createSampleData1();
828
829        XYSeries series = new XYSeries( key: "Test" );
830        for (int i = 0; i < 11; i++) {
831            series.add(data[i][0], data[i][1]);
832        }
833        XYDataset ds = new XYDatasetCollection(series);
834        double[] result = Regression.getPowerRegression(ds, series: 0);
835
836        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
837        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
838    }
839
840    /**
841     * Checks the results of a power regression on sample dataset 1 AFTER
842     * converting it to an XYSeries.
843     */
844    public void testPowerRegression10b() {
845
846        double[][] data = createSampleData1();
847
848        XYSeries series = new XYSeries( key: "Test" );
849        for (int i = 0; i < 11; i++) {
850            series.add(data[i][0], data[i][1]);
851        }
852        XYDataset ds = new XYDatasetCollection(series);
853        double[] result = Regression.getPowerRegression(ds, series: 1);
854
855        assertEquals( expected: 0.91045813, result[0], delta: 0.0000001);
856        assertEquals( expected: 0.88918346, result[1], delta: 0.0000001);
857    }
858
859    /**
860     * Checks the results of an OLS regression on sample dataset 1 AFTER

```

The screenshot shows the JFreechart 1.0.10 IDE interface with two code editors and a test results window.

Left Editor: Regression.java

```
/*
 * stage 0 - initiate errors for tests
 * try 1 - replace order sumY before sumX - work ok as should
 * try 2 - replace order sumXY before sumX - work ok as should
 * try 3 - replace order sumXX before sumX - work ok as should
 * try 4 - replace order sumXY before sumY - work ok as should
 * try 5 - change value of x - fail as should - 2 errors in tests
 * try 6 - change value of y
 */
public static double[] getPowerRegression(double[][] data) {
    int n = data.length;
    if (n < 2) {
        throw new IllegalArgumentException("Not enough data.");
    }

    double sumX = 0;
    double sumY = 0;
    double sumXX = 0;
    double sumXY = 0;
    for (int i = 0; i < n; i++) {
        double x = Math.log(data[i][0]);
        // double y = Math.log(data[i][1]);
        double y = 1;
        sumX += x;
        sumY += y;
        double xx = x * x;
        sumXX += xx;
        double xy = x * y;
        sumXY += xy;
    }
    double sxx = sumXX - (sumX * sumX) / n;
    double sxy = sumXY - (sumX * sumY) / n;
    double xbar = sumX / n;
    double ybar = sumY / n;
}
```

Run: RegressionTests > getPowerRegression()

Right Editor: RegressionTests.java

```
/*
 * Checks the results of an OLS regression on sample data
 * converting it to an XYSeries.
 */
public void testOLSRegression1b() {...}

/**
 * Checks the results of a power regression on sample data
 */
public void testPowerRegression1a() {
    double[][] data = createSampleData1();
    assertEquals( expected: 0.91045813, result[0], delta: 0.0001 );
    assertEquals( expected: 0.88918346, result[1], delta: 0.0001 );
}

/**
 * Checks the results of a power regression on sample data
 * converting it to an XYSeries.
 */
public void testPowerRegression1b() {
    double[][] data = createSampleData1();

    XYSeries series = new XYSeries( key: "Test" );
    for (int i = 0; i < 11; i++) {
        series.add(data[i][0], data[i][1]);
    }
    XYDataset ds = new XYDatasetCollection(series);
    double[] result = Regression.getPowerRegression(ds,
        1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 );

    assertEquals( expected: 0.91045813, result[0], delta: 0.0001 );
    assertEquals( expected: 0.88918346, result[1], delta: 0.0001 );
}
```

Run: RegressionTests > suite()

Bottom Left: Test Results

Test	Time	Failure Description
RegressionTests	22 ms	
org.jfree.data.statistics.junit.RegressionTests	22 ms	junit.framework.AssertionFailedError: expected:<106.1241681> but was:<2.718281828459045>
testOLSRegression1a	2 ms	<1 internal call>
testOLSRegression1b	10 ms	at junit.framework.Assert.failNotEquals(Assert.java:282) <2 internal calls> at org.jfree.data.statistics.junit.RegressionTests.testPowerRegression2(RegressionTests.java:179) at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
testPowerRegression1a	7 ms	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.base/java.lang.reflect.Method.invoke(Method.java:564) <2 internal calls>
testPowerRegression1b	0 ms	at junit.textui.TestRunner.doRun(TestRunner.java:116) <1 internal call>
testOLSRegression2a	0 ms	at junit.textui.TestRunner.doRun(TestRunner.java:109) <4 internal calls>
testOLSRegression2b	1 ms	
testPowerRegression2a	2 ms	
testPowerRegression2b	0 ms	

Process finished with exit code 255

Tests failed: 2, passed: 6 (moments ago) 205:45

האלגוריתם של loop split ושל סלידינג:

Mechanics for Split Loop

- Perform sliding on the loop fragment choosing a subset V of the loop's results.
 - *Avoid unnecessary code duplication by adding to V all loop results whose addition will not increase the size of the first resulting loop.*
 - *If sliding fails update the code to avoid the failure and repeat this step, or choose a different loop to split.*
- Compile and test.

```
ProgramSlidingOnThePDG( $N, E, n_{entry}, n_{exit}, V$ )  
1 initialize the set of edges  $FlowToExit_{NonV}$  to the empty set  
2 forall  $(m, n_{exit}, tag) \in E$  do  
3   if  $Vars(tag) \cap V = \emptyset$   
4     add the edge  $(m, n_{exit}, tag)$  to  $FlowToExit_{NonV}$   
5    $N_V := ComputeSlice(N, E \setminus FlowToExit_{NonV}, n_{exit})$   
6 initialize  $NonFinalUsesAtNode$  to map each  $n \in N$  to the empty set (of variables)  
7 forall  $(m, n, tag) \in E$  do  
8   if  $Kind(tag)$  is anti  
9     add all variables in  $Vars(tag)$  to  $NonFinalUsesAtNode(m)$   
10 initialize the set of edges  $FlowToFinalUse_V$  to the empty set  
11 forall  $(m, n, tag) \in E$  do  
12   if  $Kind(tag)$  is flow and  $Vars(tag) \subseteq (V \setminus NonFinalUsesAtNode(n))$   
13     add  $(m, n, tag)$  to the set of final-use edges  $FlowToFinalUse_V$   
14    $N_{CoV} := ComputeSlice(N, E \setminus FlowToFinalUse_V, n_{exit})$   
15    $(Pen1, Pen2, Pen3) := CollectVariablesRequiringCompensation(N, E, V,$   
16      $n_{entry}, N_V, N_{CoV}, NonFinalUsesAtNode)$   
17 return  $(N_V, N_{CoV}, Pen1, Pen2, Pen3)$ 
```

צעד מקדים לפני תחילת הריפטור:

נבצע צעד מקדים ונשנה את `for`(`while`)
ונוציא את ההגדרה של `while` - `while`.

הקוד שנקלב ושיינו נבדק נראה כך:
(כਮון שקייםנו ובדקנו שדבר לא נהרס כתוצאה לכך)

The screenshot shows an IDE interface with two main panes. The top pane displays the `Regression.java` file from the `org.jfree.statistics` package. The code implements a `getPowerRegression` method that calculates power regression coefficients. The bottom pane shows the `Run` tab with the `RegressionTests` configuration selected. The `Test Results` section indicates that 8 tests have passed in 13 ms. The list of passed tests includes various regression methods like OLS and power regression for both linear and quadratic models.

```
//change - for to while
public static double[] getPowerRegression(double[][] data) {
    int n = data.length;
    if (n < 2) {
        throw new IllegalArgumentException("Not enough data.");
    }
    double sumX = 0;
    double sumY = 0;
    double sumXX = 0;
    double sumXY = 0;
    int i = 0;
    while(i<n){
        double x = Math.log(data[i][0]);
        double y = Math.log(data[i][1]);
        sumX += x;
        sumY += y;
        double xx = x * x;
        sumXX += xx;
        double xy = x * y;
        sumXY += xy;
        i++;
    }
    double sxx = sumXX - (sumX * sumX) / n;
    double sxy = sumXY - (sumX * sumY) / n;
    double xbar = sumX / n;
    double ybar = sumY / n;
    double[] result = new double[2];
    result[1] = sxy / sxx;
    result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);
    return result;
}
```

Run: RegressionTests

Test	Time
Test Results	13 ms
org.jfree.data.statistics.junit.RegressionTests	13 ms
testOLSRegression1b	11 ms
testOLSRegression1a	0 ms
testPowerRegression1a	1 ms
testPowerRegression1b	0 ms
testOLSRegression2a	0 ms
testOLSRegression2b	0 ms
testPowerRegression2a	1 ms
testPowerRegression2b	0 ms

Tests passed: 8 (a minute ago)

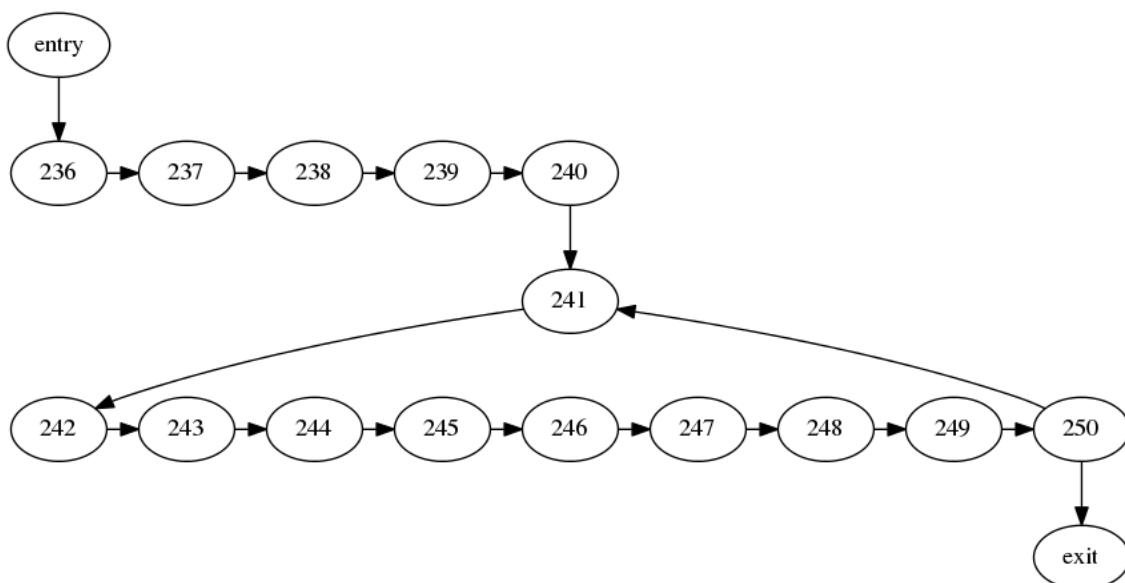
שלב 1 באלגוריתם `loop split` - ביצוע סליידינג {Xsum}

הקוד בו נתמך הוא:

```
236     double sumX = 0;  
237     double sumY = 0;  
238     double sumXX = 0;  
239     double sumXY = 0;  
240     int i = 0;  
241     while(i<n) {  
242         double x = Math.log(data[i][0]);  
243         double y = Math.log(data[i][1]);  
244         sumX += x;  
245         sumY += y;  
246         double xx = x * x;  
247         sumXX += xx;  
248         double xy = x * y;  
249         sumXY += xy;  
250         i++;  
251     }
```

ניצור את גרף ה **CFG**

*חסירה הקשת מקודקוד כניסה ליציאה



נמצא את כל הקשנות control -i data output לא קשנות הoutput

$$E_c = \{ (\text{Entry}, 236), (\text{Entry}, 237), (\text{Entry}, 238), (\text{Entry}, 239), (\text{Entry}, 240), (\text{Entry}, 241), (241, 242), (241, 243), (241, 244), (241, 245), (241, 246), (241, 247), (241, 248), (241, 249), (241, 245) \}$$

משתנה	סוג	קש	#
{ n }	flow	(Entry,241)	1
{ data.* , data }	flow	(Entry,242)	2
{ data.* , data }	flow	(Entry,243)	3
{ i }	flow	(240,241)	4
{ i }	flow	(240,242)	5
{ i }	flow	(240,243)	6
{ i }	flow	(240,250)	7
{ i }	flow	(250,240)	8
{ i }	flow	(250,250)	9
{ sumx }	flow	(236,244)	10
{ sumx }	flow	(244,244)	11
{ sumy }	flow	(237,245)	12
{ sumy }	flow	(245,245)	13
{ sumxx }	flow	(238,247)	14
{ sumxx }	flow	(247,247)	15
{ sumxy }	flow	(239,249)	16
{ sumxy }	flow	(249,249)	17
{ x }	flow	(242,244)	18
{ x }	flow	(242,246)	19
{ x }	flow	(242,248)	20
{ y }	flow	(243,245)	21
{ y }	flow	(243,249)	22
{ xx }	flow	(246,247)	23
{ xy }	flow	(248, 249)	24
{ i }	anti	(241,250)	25
{ sumx }	anti	(244,244)	26
{ sumy }	anti	(245,245)	27
{ sumxx }	anti	(247,247)	28
{ sumxy }	anti	(249,249)	29
{ i }	anti	(250,250)	30
{ n, data.* , data }	flow	(Entry,Exit)	31

{ sumx }	flow	(244,Exit)	32
{ sumy }	flow	(245,Exit)	33
{ sumxx }	flow	(247,Exit)	34
{ sumxy }	flow	(249,Exit)	35
{ i }	flow	(250,Exit)	36

cut שיש לנו את הגרפים - CFG PDG ואת ה CFG
 נתחל לבצע את אלגוריתם *sliding* (מצורף בהתחלה – נעבד לפ' השלבים שלו) נבחר תחילת עבור
 $v=\{sumx\}$

לפי שלב 4-1 באלגוריתם:

נאתחל סט שיחזיק את הצלעות $Flow To Exit_{nonV}$

$$FlowToExit_{nonV} = \{ < (\text{Entry}, \text{Exit}), \text{data}, \text{data.*}, n >$$

$$< (245, \text{Exit}), \text{sumy} >$$

$$< (247, \text{Exit}), \text{sumxx} >$$

$$< (249, \text{Exit}), \text{sumxy} >$$

$$< (250, \text{Exit}), \text{i} > \}$$

לפי שלב 5 חישוב N_V : ComputeSlice

$$N_V = Compute\ Slice\ (N, E \setminus FlowToExit_{nonV}, n_{exit})$$

נחשב את גרפ' G' המתקבל מהסרת הקשת $.FlowToExit_{nonV}$
 N_V יהיה חישוב הקודקודים של ה slice של exit לפ' G' – קלומר האבות הקדמוניים של exit הנובעים מהקשות Flow ו/או Control (כאשר נתקדם נגד כיוון השעון החל מ exit).

נשתמש באלגוריתם הבא לחישוב בסליו:

ComputeSlice(N, E, c)

- 1 initialize the result set of nodes *Slice* to the singleton set $\{c\}$
- 2 similarly initialize *Worklist* to $\{c\}$
- 3 **while** *Worklist* is not empty **do**
- 4 take the first element n out of the *Worklist*
- 5 **forall** $m \in N$ such that $(m, n, tag) \in E$ **do**
- 6 **if** *Kind(tag)* is *flow* or *control* and $m \notin Slice$
- 7 add the PDG predecessor node m of n to *Slice* and to *Worklist*
- 8 **return** *Slice*

$$N = \{Entry, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, Exit\}$$

$$FlowToExit_{nonV} = \{(Entry, Exit), (247, Exit), (249, Exit), (250, Exit), (245, Exit)\}$$

$$C = n_{exit}$$

נקבל לאחר הפעלת האלגוריתם:

$$N_V = Compute\ Slice\ (N, E \setminus FlowToExit_{nonV}, n_{exit})$$

$$N_v = \{Entry, 236, 240, 241, 242, 244, 250, Exit\}$$

שלב 6 אתחול : Non Final Uses At Node

נמצא את המשתנים שבנקודת היציאה עדין "חיים" בנ"ק היציאה .

אתחול Non Final Uses At Node={}

שלב 9-7 חישוב Non Final Uses At Node :

Non Final Uses At Node = { $<(244, sumx)>$,
 $<245, sumy>$,
 $<247, sumxx>$,
 $<249, sumxy>$,
 $<250, i>$,
 $<241, i>$ }

שלב 10 אתחול :

Flow To Final Use_V = {}

שלב 11-13

נחשב את הערך של הקשתות USE_V Flow To Final USE_V כר שאם הקשת היא flow

(n,h) והקשת אינה נוצרה בגל משטנה שנמצא ב

Non Final Uses At Node (n)

از נוסיף את הצלע ל non Final

$V = \{sumx\}$

Flow To Final Uses_V = $<(244, exit), \{sumx\}>$

אלגוריתם שלב 14:

נחשב את ה Coslice ע"י

Compute Slice (N, E \FlowToFinaluse_v, n_{exit})

Flow To Final Uses_V = $<(244, exit), \{sumx\}>$

נקבל ע"י הפעלת האלגוריתם

$N_{cov} = \{Entry, 237, 238, 239, 240, 241, 242, 243, 245, 246, 247, 248, 249, 250, Exit\}$

שלב 15-16 חישוב פיצויים

נחשב (Pen1, Pen2, Pen3)
ע"י שימוש בפונקציה

Collect Variables Requiring Compensation(N, E, V, n_{entry}, N_vN_{cov}, nonFinalusesATNode)

$$\begin{aligned}N_V &= \{Entry, 236, 240, 241, 242, 244, 250, Exit\} \\N_{cov} &= \{Entry, 237, 238, 239, 240, 241, 242, 243, 245, 246, 247, 248, 249, 250, Exit\} \\nonFinalusesATNode &= \{(244, sumx), \\&(245, sumy), (247, sumxx), (249, sumxy), (250, i)\}\end{aligned}$$

המשך חישוב פיצוי

נستخدم באלגוריתם הבא לחישוב הפיצויים

```
CollectVariablesRequiringCompensation(N, E, V,
nentry, NV, NCov, NonFinalUsesAtNode)
1 initialize Pen1, Pen2, and Pen3 to empty sets
2 forall n ∈ NCov do
3   forall v ∈ Def(n) ∩ V do
4     add the pair (n, v) to Pen1
5   forall v ∈ Use(n) ∩ V ∩ NonFinalUsesAtNode(n) do
6     add the pair (n, v) to Pen2
7 initialize InputToCoSliceNonV to the empty set
8 forall n ∈ NCov do
9   forall (nentry, n, tag) ∈ E do
10    if Kind(tag) is flow
11      add all variables in Vars(tag) \ V to InputToCoSliceNonV
12 forall n ∈ NV do
13   forall v ∈ Def(n) ∩ InputToCoSliceNonV do
14     add the pair (n, v) to Pen3
15 return (Pen1, Pen2, Pen3)
```

שלב 1 – אתחול פנים לריק

Pen1 = {}, Pen2 = {}, Pen3 = {} ,

שלב 4-2 – חישוב Pen1

כל משתנה $v \rightarrow v$, אם קיים קודקוד n שעושה define ל v .

נכnis אותו ל Pen1

קיבלנו $\{v\} = Pen$

שלב 6-5-2 – חישוב Pen2

קיים שימוש בקו סלייס של $\{v\} = Pen2$

ערך ביניים של המחולץ

שלב 7-

אתחול $inputToCoslicenonV = \{\}$

שלב 8-11

מחשב את קבוצת המשתנים שבগלם יוצאה קש特 flow מקודקוד v entry N_{cov} ו $inputToCoSliceNonV = (data, data.* , n)$

שלב 12-14

חישוב $Pen3$

Slice עלול לשנות $\{v\} = Pen3$ את ערכו של המשתנה שלא חולץ (שאנו ב- v)

ערכו ההתחלתי דרוש בקו סלייס

שלב 15

החזרת ה Pen

$Pen1 = \{v\}, Pen2 = \{v\}, Pen3 = \{v\}$

חזרה ל sliding

שלב 17 – קבלת החישוב הסופי

$N_V = \{Entry, 236, 240, 241, 242, 244, 250, Exit\}$

$N_{cov} = \{Entry, 237, 238, 239, 240, 241, 242, 243, 245, 246, 247, 248, 249, 250, Exit\}$

Pen1 = {}

Pen2 = {}

Pen3 = {}

הקוד החדש שנתקבל עבור ה(X)
slice-($v = \text{sum}X$)

```
//slice
double sumX = 0;
int i = 0;
while(i < n) {
    double x = Math.log(data[i][0]);
    sumX += x;
    i++;
}
```

הקוד החדש שנתקבל עבור ה(X)
co-slice-($v = \text{sum}X$)

```
//co slice
int i = 0;
while(i < n) {
    double x = Math.log(data[i][0]);
    double y = Math.log(data[i][1]);
    sumY += y;
    double xx = x * x;
    sumXX += xx;
    double xy = x * y;
    sumXY += xy;
    i++;
}
```

נחזיר חזרה ללואת for

נקטפל ונרי'ץ טופ'ם

The screenshot shows an IDE interface with two main panes. The top pane displays a Java code editor for a file named 'Regression.java'. The code implements a method to calculate power regression statistics from a 2D double array. The bottom pane shows the 'Run' tab with the 'RegressionTests' configuration selected. The 'Test Results' section lists eight tests under 'org.jfree.data.statistics.junit.RegressionTests', all of which have passed. The output window indicates a total execution time of 11 ms and a process exit code of 0.

```
307     /* V = {sumX}
308     after change back to for loop
309     */
310     public static double[] getPowerRegression(double[][] data) {
311         int n = data.length;
312         if (n < 2) {
313             throw new IllegalArgumentException("Not enough data.");
314         }
315         //slice
316         double sumX = 0;
317         for (int i = 0; i < n; i++) {
318             double x = Math.log(data[i][0]);
319             sumX += x;
320         }
321         //co slice
322         double sumY = 0;
323         double sumXX = 0;
324         double sumXY = 0;
325         for (int i = 0; i < n; i++) {
326             double x = Math.log(data[i][0]);
327             double y = Math.log(data[i][1]);
328             sumY += y;
329             double xx = x * x;
330             sumXX += xx;
331             double xy = x * y;
332             sumXY += xy;
333         }
334     }
```

Regression

Run: RegressionTests ×

Test	Time	Output
org.jfree.data.statistics.junit.RegressionTests	11 ms	/Library/Java/JavaVirtualMachines
testOLSRegression1a	2 ms	Process finished with exit code 0
testOLSRegression1b	8 ms	
testPowerRegression1a	0 ms	
testPowerRegression1b	0 ms	
testOLSRegression2a	0 ms	
testOLSRegression2b	0 ms	
testPowerRegression2a	1ms	
testPowerRegression2b	0 ms	

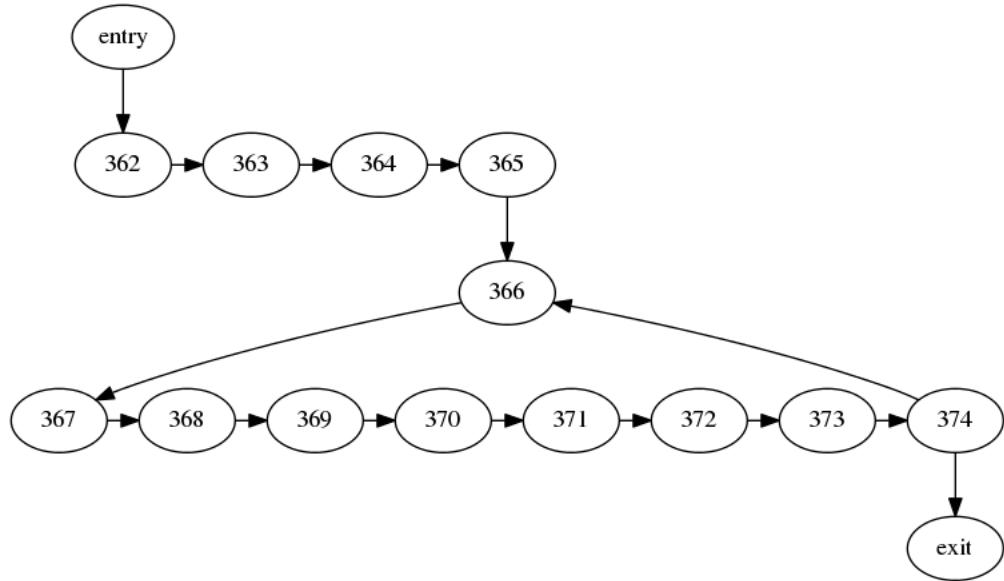
Tests passed: 8 (moments ago)

כעת נבצע שוב sliding v={sumy}

על שורות 362-374

```
351 @ public static double[] getPowerRegression(double[][] data) {  
352     int n = data.length;  
353     if (n < 2) {  
354         throw new IllegalArgumentException("Not enough data.");  
355     }  
356     double sumX = 0;  
357     for (int i = 0; i < n; i++) {  
358         double x = Math.log(data[i][0]);  
359         sumX += x;  
360     }  
361     //V = {sumY}  
362     double sumY = 0;  
363     double sumXX = 0;  
364     double sumXY = 0;  
365     int i = 0;  
366     while(i<n) {  
367         double x = Math.log(data[i][0]);  
368         double y = Math.log(data[i][1]);  
369         sumY += y;  
370         double xx = x * x;  
371         sumXX += xx;  
372         double xy = x * y;  
373         sumXY += xy;  
374         i++;  
375     }  
376     double sxx = sumXX - (sumX * sumX) / n;  
377     double sxy = sumXY - (sumX * sumY) / n;  
378     double xbar = sumX / n;  
379     double ybar = sumY / n;  
380     double[] result = new double[2];  
381     result[1] = sxy / sxx;  
382     result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);  
383     return result;  
384 }  
385 }
```

নিיצר שוב CFG



בנייה את PDG

$$\begin{aligned}
 E_c &= \\
 &= \left\{ \begin{array}{l} (\text{Entry}, 362), (\text{Entry}, 363), (\text{Entry}, 364), (\text{Entry}, 365), (\text{Entry}, 365), (\text{Entry}, 366), \\ (366, 367), (366, 368), (366, 369), (366, 370), (366, 371), (366, 372), (366, 373), (366, 374), \end{array} \right\}
 \end{aligned}$$

Data edge

משתנה	סוג	קשר	#
{ n }	flow	(Entry,366)	1
{ data, data.* }	flow	(Entry,367)	2
{ data, data.* }	flow	(Entry,368)	3
{ i }	flow	(365,366)	4
{ i }	flow	(365,367)	5
{ i }	flow	(365,368)	6
{ i }	flow	(365,374)	7
{ i }	flow	(374,374)	8
{ sumy }	flow	(362,369)	9
{ sumy }	flow	(369, 369)	10
{ sumxx }	flow	(363,371)	11
{ sumxx }	flow	(371,371)	12
{ sumxy }	flow	(364,373)	13
{ sumxy }	flow	(373, 373)	14
{ x }	flow	(367,370)	15
{ x }	flow	(367,372)	16

{y }	flow	(368,369)	17
{ y }	flow	(368,372)	18
{ xx }	flow	(370,371)	19
{ xy }	flow	(372,373)	20
{ i }	anti	(266,274)	21
{ sumy }	anti	(369,369)	22
{ sumxx }	anti	(371,371)	23
{ sumxy }	anti	(373,373)	24
	flow	(Entry, Exit)	25
{ sumy }	flow	(369, Exit)	26
{ sumxx }	flow	(371, Exit)	27
{ sumxy }	flow	(373, Exit)	28
{ i }	flow	(374, Exit)	29

v={sumy} sliding v

כעת יש CFG+PDG נוכל להפעיל את הדלבים של האלגוריתם לחישוב הסלידיניג
לפי שלבים 4-1 באלגוריתם:

$$\begin{aligned}
 FlowToExit_{nonv} = \{ & < (\text{Entry}, \text{Exit}), \text{data}, \text{data.*}, \text{n} > \\
 & < (371, \text{Exit}), \text{sumxx} > \\
 & < (373, \text{Exit}), \text{sumxy} > \\
 & < (374, \text{Exit}), \text{i} > \}
 \end{aligned}$$

לפי שלב 5 באלגוריתם:

$$Nc = \text{Compute Slice } (N, E \setminus FlowToExit_{nonv}, n_{exit})$$

$$N = \{ \text{Entry}, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, \text{Exit} \}$$

$$\text{Flow To Final Exit}_{nonv} = \{ (\text{Entry}, \text{Exit}), (371, \text{Exit}), (373, \text{Exit}), (374, \text{Exit}) \}$$

$$c = n_{exit}$$

לאחר הפעלת האלגוריתם קיבל

$$N_V = \{Entry, 362, 365, 366, 368, 369, 374, exit\}$$

שלב 9-6 חישוב NonFinalusesAtNode

$$nonFinalusesATNode = \{(369, sumy), (371, sumxx), (373, sumxy), (374, i), (366, i)\}$$

שלב 10-13 חישוב FlowTo Final uses_V

$$\text{Flow To Final Uses}_V = <(369, exit), \{sumy\}>$$

שלב 14 חישוב co-slice

$$N_{cov} = \{Entry, 362, 363, 364, 365, 366, 367, 368, 370, 371, 372, 373, 374, Exit\}$$

שלב 15-16 חישוב פיצויים :

נשתמש ב אלגוריתם לצורך חישוב הפיצויים במידה ויש כאלה

לאחר הפעלת האלגוריתם קיבל

$$\text{Pen1} = \{\}, \text{Pen2} = \{\}, \text{Pen3} = \{\}$$

שלב 17 - קבלת ערכים סופיים

$$N_V = \{Entry, 362, 365, 366, 368, 369, 374, exit\}$$

$$N_{cov} = \{Entry, 362, 363, 364, 365, 366, 367, 368, 370, 371, 372, 373, 374, Exit\}$$

$$\text{Pen1} = \{\}$$

$$\{\} \text{Pen2} =$$

$$\{\} \text{Pen3} =$$

הקוד החדש שקיבלנו עבור ה-slice

```
//slice
double sumY = 0;
int i = 0;
while(i<n) {
    double y = Math.log(data[i][1]);
    sumY += y;

    i++;
}
```

הקוד החדש שקיבלנו עבור co-slice

```
//co - slice
double sumXX = 0;
double sumXY = 0;
int i = 0;
while(i<n) {
    double x = Math.log(data[i][0]);
    double y = Math.log(data[i][1]);
    sumY += y;
    double xx = x * x;
    sumXX += xx;
    double xy = x * y;
    sumXY += xy;
    i++;
}
```

אם כן קימפלנו ורצינו את הטעות –

The screenshot shows an IDE interface with two main panes. The top pane displays a portion of a Java source code file named `Regression.java`. The code implements a method `getPowerRegression` which calculates power regression statistics from a 2D double array. The bottom pane shows the `Run` tab with a `RegressionTests` configuration selected. The test results table lists eight tests under `org.jfree.data.statistics.junit.RegressionTests`, all of which have passed. The total execution time for all tests is 10 ms.

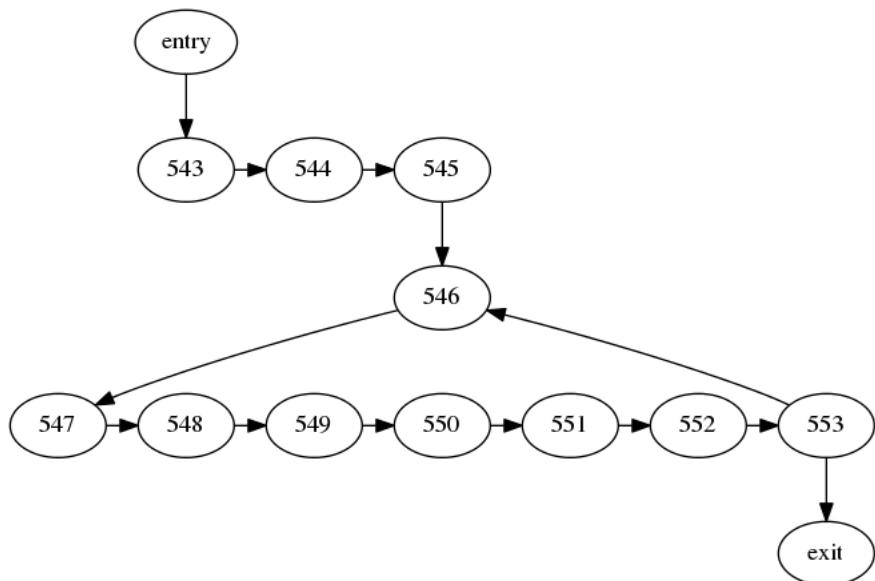
Test	Time
testOLSRegression1a	1 ms
testOLSRegression1b	9 ms
testPowerRegression1a	0 ms
testPowerRegression1b	0 ms
testOLSRegression2a	0 ms
testOLSRegression2b	0 ms
testPowerRegression2a	0 ms
testPowerRegression2b	0 ms

נמשיך ונבצע Sliding v={sumxx}

קטע הקוד בו נתמקד:

```
542 // V={sumY}
543 double sumXX = 0;
544 double sumXY = 0;
545 int i = 0;
546 while(i<n) {
547     double x = Math.log(data[i][0]);
548     double y = Math.log(data[i][1]);
549     double xx = x * x;
550     sumXX += xx;
551     double xy = x * y;
552     sumXY += xy;
553     i++;
554 }
```

תחילת ניצר CFG



נייצר PDG

$$E_c = \left\{ \begin{array}{l} ((Entry, 543), (Entry, 544), (Entry, 545), (Entry, 546), (546, 547), (546, 547),) \\ \quad (546, 548), (546, 549), (546, 550), (546, 551), (546, 552), (546, 553) \end{array} \right\}$$

data edge

משתנה	orz	קשר	#
{ data, data.* }	flow	(Entry,547)	1
{ data, data.* }	flow	(Entry,548)	2
{ n }	flow	(Entry,546)	3
{ i }	flow	(545,546)	4
{ i }	flow	(545,547)	5
{ i }	flow	(545,548)	6
{ i }	flow	(545,553)	7
{ i }	flow	(553,553)	8
{ x }	flow	(549,550)	9
{ x }	flow	(549, 552)	10
{ y }	flow	(548,552)	11
{ xx }	flow	(549,550)	12
{ sumxx }	flow	(543,550)	13
{ sumxy }	flow	(544, 553)	14
{ sumxx }	flow	(550,550)	15
{ sumxy }	flow	(552,552)	16
{i }	anti	(546,553)	17
{ sumxx }	anti	(550,550)	18
{ sumxy }	anti	(552,552)	19
{ n,data,data.* }	flow	(Entry,exit)	20
{ sumxx }	flow	(551,exit)	21
{ sumxy }	flow	(552, exit)	22
{ i }	flow	(553, exit)	23

cut שיש לנו גם את CFG וגם את PDG נוכל שוב לבצע את שלב האלגוריתם של הסלידינג על $\{sumXX = v\}$.

לפי שלבים 4-1 באלגוריתם:

$FlowToExit_{nonv} = \{ < (Entry, Exit), data, data.*, n >$
 $< (552, Exit), sumxy >$
 $< (553, Exit), i > \}$

לפי שלב 5 באלגוריתם:

$Nc = ComputeSlice(N, E \setminus FlowToExit_{nonv}, n_{exit})$
 $N = \{Entry, 547, 548, 549, 550, 551, 552, 553, Exit\}$
 $Flow To Final Exit_{nonv} = \{(Entry, Exit), (552, Exit), (553, Exit)\}$
 $c = n_{exit}$

לאחר הפעלת האלגוריתם נקבל

$N_v = \{Entry, 543, 545, 546, 547, 550, 553, Exit\}$

שלב 9-6 חישוב :NonFinalusesAtNode

$nonFinalusesATNode = \{(369, sumy), (550, sumxx), (552, sumxy), (553, i), (545, i)\}$

שלב 10-13 חישוב FlowTo Final uses_v

$Flow To Final Uses_v = < (550, exit), \{sumXX\} >$

שלב 14 חישוב co-slice

$N_{cov} = \{Entry, 544, 545, 546, 547, 548, 551, 552, 553, 554, Exit\}$

שלב 16-15 חישוב פיצויים :

נשתמש ב אלגוריתם לצורך חישוב הפיצויים במידה יש כלו

לאחר הפעלת האלגוריתם נקבל

Pen1= {}, Pen2= {}, Pen3= {}

שלב 17 - קבלת ערכים סופיים

$$N_V = \{Entry, 543, 545, 546, 547, 550, 553, Exit\}$$

$$N_{cov} = \{Entry, 544, 545, 546, 547, 548, 551, 552, 553, 554, Exit\}$$

Pen1 = {}

{}Pen2 =

{}Pen3 =

להלן הקוד הסופי שקיבלנו לאחר ביצוע sliding v={sumXX}

Slice:

```
// slice
double sumXX = 0;
int i = 0;
while(i<n) {
    double x = Math.log(data[i][0]);
    double xx = x * x;
    sumXX += xx;
    i++;
}
```

Co-slice:

```
//co slice
double sumXY = 0;
int i = 0;
while(i<n) {
    double x = Math.log(data[i][0]);
    double y = Math.log(data[i][1]);
    double xy = x * y;
    sumXY += xy;
    i++;
}
```

נחזיר את הקוד לילולאת for נק מפל ונריץ טפסים

The screenshot shows an IDE interface with two main panes. The top pane displays the `Regression.java` file, which contains Java code for calculating regression statistics. The bottom pane shows the test results for the `RegressionTests` class.

Regression.java Content:

```
622 @ ...  
623     public static double[] getPowerRegression(double[][] data) {  
624         int n = data.length;  
625         if (n < 2) {  
626             throw new IllegalArgumentException("Not enough data.");  
627         }  
628         double sumX = 0;  
629         for (int i = 0; i < n; i++) {  
630             double x = Math.log(data[i][0]);  
631             sumX += x;  
632         }  
633         double sumY = 0;  
634         for (int i = 0; i < n; i++) {  
635             double y = Math.log(data[i][1]);  
636             sumY += y;  
637         }  
638         // slice  
639         double sumXX = 0;  
640         for (int i = 0; i < n; i++) {  
641             double x = Math.log(data[i][0]);  
642             double xx = x * x;  
643             sumXX += xx;  
644         }  
645         //co slice  
646         double sumXY = 0;  
647         for (int i = 0; i < n; i++) {  
648             double x = Math.log(data[i][0]);  
649             double y = Math.log(data[i][1]);  
650             double xy = x * y;  
651             sumXY += xy;  
652         }  
653     }
```

Test Results:

Test Method	Time
<code>testOLSRegression1a</code>	2 ms
<code>testOLSRegression1b</code>	8 ms
<code>testPowerRegression1a</code>	0 ms
<code>testPowerRegression1b</code>	0 ms
<code>testOLSRegression2a</code>	0 ms
<code>testOLSRegression2b</code>	1 ms
<code>testPowerRegression2a</code>	0 ms
<code>testPowerRegression2b</code>	0 ms

Status: Tests passed: 8 (moments ago)

cut ניצא בעזרת - extract method את ה slice שקיבלנו עבור כל אחד ובכך קיבל 4 שיטות שונות לחישוב המשתנים שהילצנו

```
731     private static double getSumXY(double[][] data, int n) {  
732         double sumXY = 0;  
733         for (int i = 0; i < n; i++) {  
734             double x = Math.log(data[i][0]);  
735             double y = Math.log(data[i][1]);  
736             double xy = x * y;  
737             sumXY += xy;  
738         }  
739         return sumXY;  
740     }  
741  
742     @ private static double getSumXX(double[][] data, int n) {  
743         double sumXX = 0;  
744         for (int i = 0; i < n; i++) {  
745             double x = Math.log(data[i][0]);  
746             double xx = x * x;  
747             sumXX += xx;  
748         }  
749         return sumXX;  
750     }  
751  
752     @ private static double getSumY(double[][] data, int n) {  
753         double sumY = 0;  
754         for (int i = 0; i < n; i++) {  
755             double y = Math.log(data[i][1]);  
756             sumY += y;  
757         }  
758         return sumY;  
759     }  
760  
761     @ private static double getSumX(double[][] data, int n) {  
762         double sumX = 0;  
763         for (int i = 0; i < n; i++) {  
764             double x = Math.log(data[i][0]);  
765             sumX += x;  
766         }  
767         return sumX;  
768     }  
769 }
```

cutet נקמפל ונרייך טוטיטם:

The screenshot shows an IDE interface with the following details:

- Project Structure:** jfreechart-1.0.10 > src > org > jfree > data > statistics > Regression
- Code Editor:** Regression.java (selected tab) and RegressionTests.java.
- Code Snippet (Regression.java):**

```
707     */
708     /*
709      * Final split loop code - with Extract Method
710      */
711     @ public static double[] getPowerRegression(double[][] data) {
712         int n = data.length;
713         if (n < 2) {
714             throw new IllegalArgumentException("Not enough data.");
715         }
716         double sumX = getSumX(data, n);
717         double sumY = getSumY(data, n);
718         double sumXX = getSumXX(data, n);
719         double sumXY = getSumXY(data, n);
720         double sxx = sumXX - (sumX * sumX) / n;
721         double sxy = sumXY - (sumX * sumY) / n;
722         double xbar = sumX / n;
723         double ybar = sumY / n;
724         double[] result = new double[2];
725         result[1] = sxy / sxx;
726         result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);
727         return result;
728     }
729 }
```
- Test Results:** Tests passed: 8 of 8 tests – 28 ms
- Test Details:**

Test	Time
org.jfree.data.statistics.junit.RegressionTests	28 ms
testOLSRegression1a	4 ms
testOLSRegression1b	22 ms
testPowerRegression1a	0 ms
testPowerRegression1b	1 ms
testOLSRegression2a	0 ms
testOLSRegression2b	0 ms
testPowerRegression2a	0 ms
testPowerRegression2b	1 ms
- Environment:** /Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home
- Message:** Process finished with exit code 0

cut גבע עבור כל אחת מהשיטות החדש שיצרנו לחישוב כל אחד מ 4 המשתנים - **'Inline temp'**
cut נקמפל ונריץ טוטים:

The screenshot shows an IDE interface with the following details:

- Code Editor:** Displays Java code for a `getPowerRegression` method. The code calculates regression statistics based on input data. It includes imports for `IllegalArgumentException`, `Math`, and various helper methods like `getSumXX`, `getSumXY`, `getSumX`, and `getSumY`.
- Run Tab:** Shows a successful run of `RegressionTests`. The status message indicates "Tests passed: 8 of 8 tests – 30 ms".
- Test Results:** A detailed list of test cases under the `Test Results` section. All tests are marked with a green checkmark, indicating they passed. The tests are:
 - `testOLSRegression1a`: 3 ms
 - `testOLSRegression1b`: 25 ms
 - `testPowerRegression1a`: 1 ms
 - `testPowerRegression1b`: 0 ms
 - `testOLSRegression2a`: 0 ms
 - `testOLSRegression2b`: 1 ms
 - `testPowerRegression2a`: 0 ms
 - `testPowerRegression2b`: 0 ms
- Output:** Shows the command used to run the tests: `/Library/Java/JavaVirtualMachines/jdk-10.0.1.jdk/Contents/Home/bin/java -jar target/Regression-1.0-SNAPSHOT.jar` and the message "Process finished with exit code 0".

לסיכון

בדוגמה זו רأינו תחילת הפעלה של `loop split` ע"י שימוש ב `sliding` לאחר מכן מכון ביצענו `temp extract method` ולאחר מכן מכון ביצענו `Inline temp` על המשתנים שנותרו קיבלנו פונקציה קצרה מושלמת ועל זאת נרחיב בדין על האם היה מושלם

דין

בדוגמה זאת נוכל לראות כי ביצוע של ה- `sliding` יצר הפרדה ל 4 לולאות שונות שכל אחת מחשבת משתנה אחר שהילצנו.

בדוגמה זו ניסינו להראות את החזקה של אלגוריתם הסליידינג על פני אלגוריתם הבקטיניג. בשונה מבקטיניג אלגוריתם הסליידינג יכול ל"פרק" חתיכות קוד מתוך לולאה ولكن במטרה כמו שלנו כשי לולאה שמחשבת 4 משתנים שונים ונרצה להפרידן אלגוריתם הסליידינג הוא זה שיוכל לבצע זאת בצורה מיטבית.

יתרונות ביצוע ה- `sliding`

- יש כתע 4 לולאות מופרדות שכל אחת מחשבת משתנה אחד בלבד בצורה זו במידה ונרצה לשנות בעתיד את חישוב המשתנה נוכל לבצע שינויים כאלו בקלות ללא חשש שנפגע במכונותם של האחרים
- כתע אנו יכולים לחלץ את כל אחת מהשיטות לפונקציה נפרדת ע"י שימוש ב `extract method` היות וראינו כי בחלוקת ישנו כפליות שمبرעות אוד זהה ובכך נוכל לחסוך בכוח חישוב ובפעילותויות מיותרות והקוד ההפוך גנרי יותר ויחסור כפליות בקוד
- הקוד ההפוך לקריא יותר מהבינה שנייה להבין אותו בקלות יתרה וכייד מחשבים כל אחד מהמשתנים
- נוכל בעתיד לשנות את אופן החישוב של כל אחד מהמשתנים שיצאנו ללא חשש שאנו הורסים קוד אחר
- כדי שציגנו חוזקתו של הסליידינג היא זאת שאיפשרה את פיצול לולאה.

חסרוןות

- הקוד שנוצר ארוך יותר
- קיבלנו 4 לולאות במקום 1 – במידה והביצועים חשובים לנו – יכול להיות שלא שווה להשאיר את הקוד הנ"ל כפי שהוא ולהחזירו לקוד המקורי
- אנו מחשבים את הערכים של x ו y מספר פעמים דבר ששוב מבחינת ביצועים יכול לעלות לנו.
- יש בחלוקת מספר clone של חלקים מהפונקציה שכפי שאמרנו נוכל לחלץ לתתי פונקציות – אך נדרש לשקל מה tradeoff של הדבר והאם כדאי שיהיה קוד נקי יותר ומסודר לאחר חילוץ הפונקציות בעלותו של הרבה פונקציות קטנות אחרות בחלוקת שואלי יהיו בשימוש רק פעם אחת.

מסקנה:

לדעתם השימוש נוחץ לאחר שהבחנו בכמות שכפולי הקוד בדרך זו נוכל לחלץ את השיטות לפונקציה גנרטית וכן לגרום לחסכוון משמעותי בשכפולי הקוד וכן הקוד עצמו של הפונקציות הפורק ליותר קרייא כਮובן שכפי שאמרנו במידה והביצועים של הפונקציה חשובים הריפורטור הנ"ל ישנה את הייעילות לשילילה החולשה של האלגוריתם הנ"ל היא שהפיצול כל פעם לפי 7 אחד שגורם לשכפול רב של קוד שיכל להיות שבगלל שיש 3^{n+1} בחלוקת עדין יהיה שווה את זה מהבחינה שפונקציות נוספות ישתמשו בקוד הנ"ל.

סביר גם למה בחרנו לבצע sliding-slides ולא bucketing

היות והקודדים שנרצה לחלץ במידה והינו מפעילים bucketing כדי להיפטר מה 3^{n+1} שיש לנו בחלוקת הינו רוצים לחלץ את כל השורות פרט ל x ו y אשר נמצאות בתוך לולאה וכיון שישנו שורות בלולאה שהן מסומנות נקלט מצב שככל הלולאה תיכנס באופן אוטומטי markerd וזו לא עשינו שום דבר בהפעלת האלגוריתם על הפונקציה הזאת כי כל הקוד היה נשאר כמקרה אחד. ביצוע sliding על משתנה בודד כל פעם אפשר לנו לחלץ את החישוב של כל משנה גם אם "עליה" לנו בביטויים וביצירת לולאות נוספות בדוגמה הבאה נדגים בדיק זאת

Type 3 – Clone Elimination – Bucketing (fail)

כעת כדי להדגים את מה שכבר טענו כלפי ה- bucketing נבצע עבור קוד המקור לפני שהפעלונו עליו את האלגוריתם split loop

נבחן כי בין השיטה `getPowerRegression(double[] data)` לבין השיטה `getPowerRegression(xydata, data, int series)` כאשר הדבר השונה ביניהן הוא אופן חישוב y , x .

יש נסתכל על השיטות ונבחים כי קיימ-3

נעביר מול הקוד הבא שכבר ראיינו:

```
236     double sumX = 0;
237     double sumY = 0;
238     double sumXX = 0;
239     double sumXY = 0;
240     int i = 0;
241     while(i<n) {
242         double x = Math.log(data[i][0]);
243         double y = Math.log(data[i][1]);
244         sumX += x;
245         sumY += y;
246         double xx = x * x;
247         sumXX += xx;
248         double xy = x * y;
249         sumXY += xy;
250         i++;
251     }
```

נסמן את השורותizzaות בטור הלוילא ואת הגדרות של המשתנים לפני

M = {236, 244}

יש לנו כבר את גראף ה CFG ואת גראף ה PDG + הקשתות החדשות הoutput :
 קשתו היקונטROL נשארו זהות גם הCFG – היות לנו ממשיכים על אותו קוד מהדוגמה הקודמת

מספר	קשת	סוג	משתנה
1	(Entry,241)	flow	{ n }
2	(Entry,242)	flow	{ data.* , data }
3	(Entry,243)	flow	{ data.* , data }
4	(240,241)	flow	{ i }
5	(240,242)	flow	{ i }
6	(240,243)	flow	{ i }
7	(240,250)	flow	{ i }
8	(250,240)	flow	{ i }
9	(250,250)	flow	{ i }
10	(236,244)	flow	{ sumx }
11	(244,244)	flow	{ sumx }
12	(237,245)	flow	{ sumy }
13	(245,245)	flow	{ sumy }
14	(238,247)	flow	{ sumxx }
15	(247,247)	flow	{ sumxx }
16	(239,249)	flow	{ sumxy }
17	(249,249)	flow	{ sumxy }
18	(242,244)	flow	{ x }
19	(242,246)	flow	{ x }
20	(242,248)	flow	{ x }
21	(243,245)	flow	{ y }
22	(243,249)	flow	{ y }
23	(246,247)	flow	{ xx }
24	(248, 249)	flow	{ xy }
25	(241,250)	anti	{ i }
26	(244,244)	anti	{ sumx }
27	(245,245)	anti	{ sumy }
28	(247,247)	anti	{ sumxx }
29	(249,249)	anti	{ sumxy }
30	(250,250)	anti	{ i }
31	(Entry,Exit)	flow	{ n, data.* , data }
32	(244,Exit)	flow	{ sumx }
33	(245,Exit)	flow	{ sumy }
34	(247,Exit)	flow	{ sumxx }
35	(249,Exit)	flow	{ sumxy }
36	(250,Exit)	flow	{ i }
37	(236,244)	output	{ sumx }
38	(237,245)	output	{ sumy }

{ sumxx }	output	(238,247)	39
{ sumxy }	output	(239,249)	40
{ i }	output	(240,250)	41
{ i }	output	(250,250)	42

בנייה את גרף slideDG

נחשב את $slideDep$

נזכיר אם קיימת קשת 'm' ל'k' קודקוד '

ששיכת ל'DataDep' וגם

$$m' \in slide(m) \text{ וגם } n' \in slide(n)$$

از מתקדים (n,m) $slideDep = (n, m)$

תזכורת:

(n) – קבוצת צמתים המכילה את n ואת כל האבות הקדמוניים שלו המקיימים בקשרות Control .

נחשב עבור כל אחד מרהוקודים תחיליה מהוא ה (n) Slide

ולאחר מכן נמצא את כל הקשרות slide depended

Slide (236) = {236,Entry}

Slide (237) = {237,Entry}

Slide (238) = {238,Entry}

Slide (239) = {239,Entry}

Slide (240) = {240,Entry}

Slide (242) = {242,243,244,245,246,247,248,249,250}

Slide (243) = {242,243,244,245,246,247,248,249,250}

Slide (244) = {242,243,244,245,246,247,248,249,250}

Slide (245) = {242,243,244,245,246,247,248,249,250}

Slide (246) = {242,243,244,245,246,247,248,249,250}

Slide (247) = {242,243,244,245,246,247,248,249,250}

Slide (248) = {242,243,244,245,246,247,248,249,250}

Slide (249) = {242,243,244,245,246,247,248,249,250}

Slide (250) = {242,243,244,245,246,247,248,249,250}

קשתות של slide DG

$$E_{SD} = \left\{ (241,241), (241,242), (241,243), (241,244), (241,245), (241,246), (241,247), (241,248), (241,249), (241,250) \right. \\ \left. (242,241), (242,242), (242,243), (242,244), (242,245), (242,246), (242,247), (242,248), (242,249), (242,250) \right. \\ \left. (243,241), (243,242), (243,243), (243,244), (243,245), (243,246), (243,247), (243,248), (243,249), (243,250) \right. \\ \left. (244,241), (244,242), (244,243), (244,244), (244,245), (244,246), (244,247), (244,248), (244,249), (244,250) \right. \\ \left. (245,241), (245,242), (245,243), (245,244), (245,245), (245,246), (245,247), (245,248), (245,249), (245,250) \right. \\ \left. (246,241), (246,242), (246,243), (246,244), (246,245), (246,246), (246,247), (246,248), (246,249), (246,250) \right. \\ \left. (247,241), (247,242), (247,243), (247,244), (247,245), (247,246), (247,247), (247,248), (247,249), (247,250) \right. \\ \left. (248,241), (248,242), (248,243), (248,244), (248,245), (248,246), (248,247), (248,248), (248,249), (248,250) \right. \\ \left. (249,241), (249,242), (249,243), (249,244), (249,245), (249,246), (249,247), (249,248), (249,249), (249,250) \right. \\ \left. (250,241), (250,242), (250,243), (250,244), (250,245), (250,246), (250,247), (250,248), (250,249), (250,250) \right. \\ \left. (236,241), (236,242), (236,243), (236,244), (236,245), (236,246), (236,247), (236,248), (236,249), (236,250) \right. \\ \left. (237,241), (237,242), (237,243), (237,244), (237,245), (237,246), (237,247), (237,248), (237,249), (237,250) \right. \\ \left. (238,241), (238,242), (238,243), (238,244), (238,245), (238,246), (238,247), (238,248), (238,249), (238,250) \right. \\ \left. (239,241), (239,242), (239,243), (239,244), (239,245), (239,246), (239,247), (239,248), (239,249), (239,250) \right\}$$

כעת יש לנו את כל הקשרים של ה slideDG

*לא נציג את הגרף מכיוון שמספר קשתות הוא לא קרייא

slide-based-bucketing(N, E_C, E_D, M)

- 1: $M\text{-reachable} := \text{slides-first-search}(N, E_C, E_D, M, \text{"forward"})$
- 2: $\text{reaching-}M := \text{slides-first-search}(N, E_C, E_D, M, \text{"backward"})$
- 3: $\text{marked} := \text{reaching-}M \cap M\text{-reachable}$
- 4: $\text{before} := \text{reaching-}M \setminus M\text{-reachable}$
- 5: $\text{after} := M\text{-reachable} \setminus \text{reaching-}M$
- 6: **return** (before,marked,after)

slides-first-search($N, E_C, E_D, M, \text{direction}$)

- 7: initialize visited-up and visited-down to \emptyset
- 8: initialize worklist and reached to M
- 9: **while** worklist is not empty **do**
- 10: take the first node m out of the worklist
- 11: newly-visited-slides := slide-dependence-search(m)
- 12: add all newly-visited-slides to reached and to the worklist
- 13: **return** reached

slide-dependence-search(m)

- 14: initialize newly-reached-slides to \emptyset
- 15: newly-visited-slide-nodes := visit-slide-of(m)
- 16: **forall** $m' \in$ newly-visited-slide-nodes **do**
- 17: **if** the direction is "forward"
- 18: **forall** $n \in N$ such that $(m', n) \in E_D$ **do**
- 19: newly-reached-slides :=
 newly-reached-slides \cup search-slides-with(n)
- 20: **else** (*i.e.* the direction is "backward")
- 21: **forall** $l \in N$ such that $(l, m') \in E_D$ **do**
- 22: newly-reached-slides :=
 newly-reached-slides \cup search-slides-with(l)
- 23: **return** newly-reached-slides

visit-slide-of(n)

24: initialize the local set of newly-visited nodes to \emptyset
25: **if** $n \notin$ visited-up
26: add n to visited-up and to newly-visited
27: **forall** $p \in N$ such that $(p,n) \in E_C$ **do**
28: newly-visited := newly-visited \cup visit-slide-of(p)
29: **return** newly-visited

search-slides-with(m)

30: initialize the local set of newly-reached nodes to \emptyset
31: **if** $m \notin$ visited-down
32: add m to visited-down
33: **if** there exists no $n \in N$ such that $(m,n) \in E_C$
34: **if** $m \notin$ reached add m to newly-reached
35: **else forall** $n \in N$ such that $(m,n) \in E_C$ **do**
36: newly-reached := newly-reached \cup search-slides-with(n)
37: **return** newly-reached

Fig. 5: Slide-based bucketing algorithm

שלב 1 נמצא את M-reachable

ע"י ביצוע האלגוריתם Slides First Search

בוצע את האלגוריתם:

נתחול:

1. Visit-up = {}
2. Work list = {236,244}

cut עבור כל אחד מהקודקודים ב work list

נפעיל את האלגוריתם SlideDependenc-Search

האלגוריתם הנ"ל מוחזיר עבור כל אחד מהקודקודים את קבוצות קודקודים אליו ניתן להגעה לפי חיפוש של כיוון התקדמות forward/backward מבצע זאת ע"י מעבר על הקשתות לפי כיוון בגרף slideDG ובודק לאילו קודקודים אנו יכולים להגיע.

האלגוריתם משתמש באלגוריתם הבא לביצוע המשימה הנ"ל

לאחר שנתקבל מ SlideDependenc-Search

את הקודקודים הוא מוחזיר אותם Slide first search

שהרץ עבור כל אחד מהקודקודים ושמր את הקבוצה אליה הגיעו

כלומר קיבלנו את $M\text{-reachable}$ שהוא קבוצת כל הקודקודים שניתן להגעה אליהם
מקודקודים M בגרף

- נזכיר $\{236,244\}$

$M\text{-reachable} = \{236,237,238,239,240,241,242,243,244,245,246,247,248,249,250\}$

שלב 2 חישוב reaching-M

באופן זהה לחישוב $M\text{-reachable}$ נחשב מהן קבוצת הקודקודים מהן ניתן להגעה אל קודודי M (צעידה אחורה) על הקשתות

-נקבל-

$Reaching-M = \{236,237,238,239,240,241,242,243,244,245,246,247,248,249,250\}$

שלב 3 חישוב Marked

$$Marked = reaching - M \cap M - reachable$$

$$Marked = \{236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250\}$$

$$before = \{ \}$$

$$after = \{ \}$$

קודקודי ה $Marked$ מצינים את קבוצת הקודקודים שאינם ניתנים להזזה •

שלב 4 חישוב before

$$before = reaching - M / M - reachable = \{ \}$$

שלב 5 חישוב after

$$after = reaching - M / M - reachable = \{ \}$$

סיכון הדוגמא:

מטרת הפעלה של slide based bucketing היא להראות את חולשת האלגוריתם הנ"ל אל מול האלגוריתם **sliding**

כפי שחזינו בדוגנוב הקודמת בנויגוד ל sliding algorithm ה剩יאר את כל הקטע קוד המבוקש שלנו כמקרה אחד. ולא הצליח לחלק את קטע הקוד המבוקש שלנו.

בדוגמא זו הצלחנו להראות כיצד מבצעים את האלגוריתם **slide** תוך מתן דגש לחולשות העיקריות שהוא אינו יכול לחלק קטע קוד המכיל קוד מתוך ומוחוץ לוואה.

מסקנות

כפי שציינו עבור קוד אשר מכיל לוואה ונרצה לחלק קוד מוחוץ ומתוך הלולאה האלגוריתם יחזיר את כל הקוד בשלהמו

Type 3 Clone elimination based in bucketing +sliding example (fail)

עבור אותו פונקציית שניסינו מקודם לבצע bucketing ללא הצלחה נבצע חתך שוב רק על קטע קוד אחר שלא כולל לולאה.

נראה כאן מהי חוזקתו של האלגוריתם הנל"ל בנויגוד לשידוק

במחלקה regression קטע הקוד חוזר 4 פעמים

```

71    double sumX = 0;
72    double sumY = 0;
73    for (int i = 0; i < n; i++) {
74        double x = data[i][0];
75        double y = data[i][1];
76        sumX += x;
77        sumY += y;
78        double xx = x * x;
79        sumXX += xx;
80        double xy = x * y;
81        sumXY += xy;
82    }
83    double sxx = sumXX - (sumX * sumX) / n;
84    double sxy = sumXY - (sumX * sumY) / n;
85    double xbar = sumX / n;
86    double ybar = sumY / n;
87
88    double[] result = new double[2];
89    result[1] = sxy / sxx;
90    result[0] = ybar - result[1] * xbar;
91
92    return result;
93}

```

```

116    double sumX = 0;
117    for (int i = 0; i < n; i++) {
118        double x = data.getValue(series, i);
119        double y = data.getYValue(series, i);
120        sumX += x;
121        sumY += y;
122        double xx = x * x;
123        sumXX += xx;
124        double xy = x * y;
125        sumXY += xy;
126    }
127    double sxx = sumXX - (sumX * sumX) / n;
128    double sxy = sumXY - (sumX * sumY) / n;
129    double xbar = sumX / n;
130    double ybar = sumY / n;
131
132    double[] result = new double[2];
133    result[1] = sxy / sxx;
134    result[0] = ybar - result[1] * xbar;
135
136    return result;
137}

```

```

189    double sumX = 0;
190    double sumY = 0;
191    double sumXX = 0;
192    double sumXY = 0;
193    for (int i = 0; i < n; i++) {
194        double x = Math.log(data.getValue(series, i));
195        double y = Math.log(data.getYValue(series, i));
196        sumX += x;
197        sumY += y;
198        double xx = x * x;
199        sumXX += xx;
200        double xy = x * y;
201        sumXY += xy;
202    }
203    double sxx = sumXX - (sumX * sumX) / n;
204    double sxy = sumXY - (sumX * sumY) / n;
205    double xbar = sumX / n;
206    double ybar = sumY / n;
207
208    double[] result = new double[2];
209    result[1] = sxy / sxx;
210    result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);
211
212    return result;
213}

```

כasher ההבדל ביןיהן היא אופן חישוב `result[0]`

וכameron חישוב ערך הראנו שי אפשר בעזרה bucketing לבצע ביטול של הcpfilioות
לכן נתמוך בקע קוד הבא אותו נרצה לבטל את הcpfilioות שלו .

פונקציה מקורית:

```
/*
 * Returns the parameters 'a' and 'b' for an equation  $y = ax^b$ , fitted to
 * the data using a power regression equation. The result is returned as
 * an array, where double[0] --> a, and double[1] --> b.
 *
 * @param data the data.
 * @param series the series to fit the regression line against.
 *
 * @return The parameters.
 */
public static double[] getPowerRegression(XYDataset data, int series) {

    int n = data.getItemCount(series);
    if (n < 2) {
        throw new IllegalArgumentException("Not enough data.");
    }
    double sumX = 0;
    double sumY = 0;
    double sumXX = 0;
    double sumXY = 0;
    for (int i = 0; i < n; i++) {
        double x = Math.log(data.getXValue(series, i));
        double y = Math.log(data.getYValue(series, i));
        sumX += x;
        sumY += y;
        double xx = x * x;
        sumXX += xx;
        double xy = x * y;
        sumXY += xy;
    }
    double sxx = sumXX - (sumX * sumX) / n;
    double sxy = sumXY - (sumX * sumY) / n;
    double xbar = sumX / n;
    double ybar = sumY / n;

    double[] result = new double[2];
    result[1] = sxy / sxx;
    result[0] = Math.pow(Math.exp(1.0), ybar - result[1] * xbar);

    return result;
}
```

ראשית נבצע עיבוד מקדים של הפונקציה היות ואופן חישוב result[0] שונה נרצה להפריד ולשמור את הערכים שנשмарים ל result[1],result[0]

לאחר עיבוד מקדים:

```
838 //original - code elimination - change result to temp
839 @    public static double[] getPowerRegression(XYDataset data, int series) {
840
841     int n = data.getItemCount(series);
842     if (n < 2) {
843         throw new IllegalArgumentException("Not enough data.");
844     }
845     double sumX = 0;
846     double sumY = 0;
847     double sumXX = 0;
848     double sumXY = 0;
849     for (int i = 0; i < n; i++) {
850         double x = Math.log(data.getXValue(series, i));
851         double y = Math.log(data.getYValue(series, i));
852         sumX += x;
853         sumY += y;
854         double xx = x * x;
855         sumXX += xx;
856         double xy = x * y;
857         sumXY += xy;
858     }
859     double sxx = sumXX - (sumX * sumX) / n;
860     double sxy = sumXY - (sumX * sumY) / n;
861     double xbar = sumX / n;
862     double ybar = sumY / n;
863     double res1 = sxy / sxx;
864     double res2 = Math.pow(Math.exp(1.0), ybar - res1 * xbar);
865     double[] result = new double[2];
866     result[1] = res1;
867     result[0] = res2;
868     return result;
869
870 }
```

אנו נעבד על פונקציה נתמוך בקטע הקוד :

```
859     double sxx = sumXX - (sumX * sumX) / n;
860     double sxy = sumXY - (sumX * sumY) / n;
861     double xbar = sumX / n;
862     double ybar = sumY / n;
863     double res1 = sxy / sxx;
864     double res2 = Math.pow(Math.exp(1.0), ybar - res1 * xbar);
```

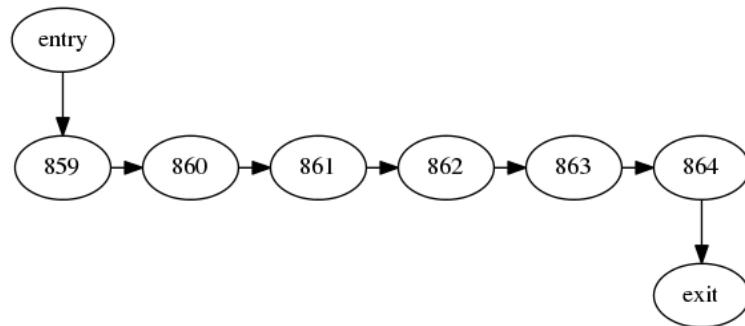
המטרה

לייצא את חישוב res כລומר את כל הקוד המשוכפל הזהה 859-863 היה ומשוכפל במחלקה 4 פעמים
ונתמקד בשורות 859-864

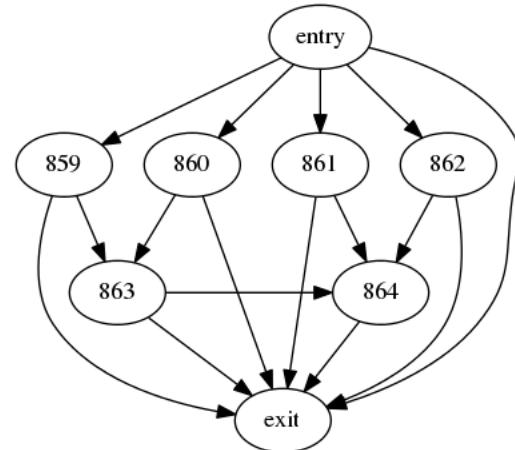
זכור משתנים המגיעים אליו מנוק' כניסה:

$$Var_{entry} = \{sumx, sumy, sumxy, sumxx\}$$

בנייה את CFG



עת נבנה את גרף ה PDG



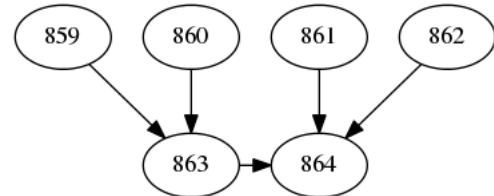
קשותות control

$$E_c = \{(Entry, 859), (Entry, 860), (Entry, 861), (Entry, 862), (Entry, 863), (Entry, 864), (Entry, Exit)\}$$

nochshav keshutot data

מספר	קשת	סוג	משתנה
1	(Entry,859)	flow	{ sumxx, sumx, n }
2	(Entry,860)	flow	{ sumxy, sumx, sumy ,n }
3	(Entry,861)	flow	{ sumx, n }
4	(Entry,862)	flow	{ sumy, n }
5	(859,863)	flow	{ sxx }
6	(866,863)	flow	{ syy }
7	(862,864)	flow	{ ybar }
8	(861,864)	flow	{ xbar }
9	(863,864)	flow	{ res1 }
10	(859,Exit)	flow	{ sxx }
11	(860, Exit)	flow	{ sxy }
12	(861, Exit)	flow	{ xbar }
13	(862, Exit)	flow	{ ybar }
14	(863, Exit)	flow	{ res1}
15	(864, Exit)	flow	{ res2 }

SlideDG הגרף



nochshavuber كل אחד מהקודדים (n) -

Slide (859)=(859,Entry)

Slide (860)=(860,Entry)

Slide (861)=(861,Entry)

Slide (862)=(862,Entry)

Slide (863)=(863,Entry)

Slide (864)=(864,Entry)

עכשו שיש לנו את כל הגרפים

סימנו את השורות {859,860,863} = M

נפעיל את האלגוריתם sliding based bucketing - מצורף בדוגמה הקודמת

שלב 1

נמצא את *M-reachable*

שהיא קבוצת השורות אליה ניתן להגיע מוקודקיי M בגרף מעבר קשთות קדימה

$$M\text{-reachable} = \{859, 860, 863, 864\}$$

שלב 2

נמצא את *M-r* קבוצת השורות שניית להגעה מהם אל M מעבר על הקשთות אחרת

$$r\text{-}M = \{859, 860, 863\}$$

שלב 3

נחשב את *Marked*

$$Marked = r - M \cap M - r = \{859, 860, 863\} \cap \{859, 860, 863, 864\} = \{859, 860, 863\} = M$$

שלב 4

חישוב קבוצת ה *before*

$$Before = r - M \setminus M - r = \{859, 860, 863, 859\} \setminus \{859, 860, 863, 864\} = \{\}$$

קיבילתנו שקבוצת ה *before* ריקה

שלב 5

חישוב קבוצת *after*

$$After = M - r \setminus r - M = \{859, 860, 863, 864\} \setminus \{859, 860, 863\} = \{864\}$$

קיבילנו כי שורה 864 חייבת להיות ממוקמת אחרי קבוצת השורות שסימנו M

שלב 6

נחזיר את *before,Marked,after*

האלגוריתם שאותו חישבנו את ה*hzir* לנו

Before ={}

Marked={859,860,863}

After={864}

- שורות 861 ו 862 יכולות להיות ממוקמות גם ב *before* וגם ב *after*

נמיקם ב *after*. באופן שרירותי כדי ליצר קוד אחד

הקוד שנקלל לאחר הפעלת ה *bucketing*

סדר השורות:

Before:

Marked: 859

860

863

After:861

862

864

```
859     double sxx = sumXX - (sumX * sumX) / n;
860     double sxy = sumXY - (sumX * sumY) / n;
861     double res1 = sxy / sxx;
862     double xbar = sumX / n;
863     double ybar = sumY / n;
864     double res2 = Math.pow(Math.exp(1.0), ybar - res1 * xbar);
865 }
```

- נסדר את הקוד החדש נקمل ונוירץ טסטים שכבר באלגוריתם הראשון עברו את הניסויות שלמו להפיל אותו לכן מידה והקוד לא עובד ויירוס משוה נגלה זאת

The screenshot shows an IDE interface with two tabs open: `Regression.java` and `RegressionTests.java`. The left pane displays the project structure for `jfreechart-1.0.10`, specifically the `src/org/jfree/statistics` package. The right pane shows the code for `Regression.java` and its test results.

```

873     //result bucketing
874     public static double[] getPowerRegression(XYDataset data, int series) {
875         @
876             int n = data.getItemCount(series);
877             if (n < 2) {
878                 throw new IllegalArgumentException("Not enough data.");
879             }
880             double sumX = 0;
881             double sumY = 0;
882             double sumXX = 0;
883             double sumXY = 0;
884             for (int i = 0; i < n; i++) {
885                 double x = Math.log(data.getXValue(series, i));
886                 double y = Math.log(data.getYValue(series, i));
887                 sumX += x;
888                 sumY += y;
889                 double xx = x * x;
890                 sumXX += xx;
891                 double xy = x * y;
892                 sumXY += xy;
893             }
894             double sxx = sumXX - (sumX * sumX) / n;
895             double sxy = sumXY - (sumX * sumY) / n;
896             double res1 = sxy / sxx;
897             double xbar = sumX / n;
898             double ybar = sumY / n;
899             double res2 = Math.pow(Math.exp(1.0), ybar - res1 * xbar);
900             double[] result = new double[2];
901             result[1] = res1;
902             result[0] = res2;
903             return result;
904     }
905 }
```

The `RegressionTests.java` tab shows the following test results:

Test Method	Time (ms)	Status
<code>testOLSRegression1a</code>	3 ms	Passed
<code>testOLSRegression1b</code>	18 ms	Passed
<code>testPowerRegression1a</code>	0 ms	Passed
<code>testPowerRegression1b</code>	0 ms	Passed
<code>testOLSRegression2a</code>	0 ms	Passed
<code>testOLSRegression2b</code>	0 ms	Passed
<code>testPowerRegression2a</code>	0 ms	Passed
<code>testPowerRegression2b</code>	0 ms	Passed

Total time: 21 ms. All 8 tests passed.

מבצע Extract Method

על השורות שקבענו. לקראת החלפת כל קטעי הקוד שמצאו במחלקה.
נكمפל ונՐץ כדי לראות שלא הרסנו כלום במהלך חילוץ השיטה החדשה זו

The screenshot shows an IDE interface with two tabs: `Regression.java` and `RegressionTests.java`. The `Regression.java` tab displays the following code:

```
910     //result bucketing - extract method
911     @ ...
912     public static double[] getPowerRegression(XYDataset data, int series) {
913         int n = data.getItemCount(series);
914         if (n < 2) {
915             throw new IllegalArgumentException("Not enough data.");
916         }
917         double sumX = 0;
918         double sumY = 0;
919         double sumXX = 0;
920         double sumXY = 0;
921         for (int i = 0; i < n; i++) {
922             double x = Math.log(data.getXValue(series, i));
923             double y = Math.log(data.getYValue(series, i));
924             sumX += x;
925             sumY += y;
926             double xx = x * x;
927             sumXX += xx;
928             double xy = x * y;
929             sumXY += xy;
930         }
931         double res1 = getRes1(n, sumX, sumY, sumXX, sumXY);
932         double xbar = sumX / n;
933         double ybar = sumY / n;
934         double res2 = Math.pow(Math.exp(1.0), ybar - res1 * xbar);
935         double[] result = new double[2];
936         result[1] = res1;
937         result[0] = res2;
938         return result;
939     }
940     @ ...
941     private static double getRes1(int n, double sumX, double sumY, double sumXX, double sumXY) {
942         double sxx = sumXX - (sumX * sumX) / n;
943         double sxy = sumXY - (sumX * sumY) / n;
944         return sxy / sxx;
945     }
946 }
```

The `RegressionTests.java` tab shows the following test results:

Test	Time	Status
testOLSRegression1a	4 ms	Passed
testOLSRegression1b	10 ms	Passed
testPowerRegression1a	1 ms	Passed
testPowerRegression1b	0 ms	Passed
testOLSRegression2a	0 ms	Passed
testOLSRegression2b	0 ms	Passed
testPowerRegression2a	0 ms	Passed
testPowerRegression2b	1 ms	Passed

Overall, 8 tests passed in 16 ms.

מבצע After על הקוד - וnochלץ שיטה עבורי

נקומפל ונרייז טווטים

The screenshot shows an IDE interface with two tabs: `Regression.java` and `RegressionTests.java`. The `Regression.java` tab displays the following code:

```
//result bucketing - extract methods |
public static double[] getPowerRegression(XYDataset data, int series) {
    int n = data.getItemCount(series);
    if (n < 2) {
        throw new IllegalArgumentException("Not enough data.");
    }
    double sumX = 0;
    double sumY = 0;
    double sumXX = 0;
    double sumXY = 0;
    for (int i = 0; i < n; i++) {
        double x = Math.log(data.getXValue(series, i));
        double y = Math.log(data.getYValue(series, i));
        sumX += x;
        sumY += y;
        double xx = x * x;
        sumXX += xx;
        double xy = x * y;
        sumXY += xy;
    }
    double res1 = getRes1(n, sumX, sumY, sumXX, sumXY);
    double res2 = getRes2Regression(n, sumX, sumY, res1);
    double[] result = new double[2];
    result[1] = res1;
    result[0] = res2;
    return result;
}

private static double getRes2Regression(int n, double sumX, double sumY, double res1) {
    double xbar = sumX / n;
    double ybar = sumY / n;
    return Math.pow(Math.exp(1.0), ybar - res1 * xbar);
}
```

The `RegressionTests.java` tab shows the following test results:

Test	Time	Result
testOLSRegression1a	2 ms	Passed
testOLSRegression1b	8 ms	Passed
testPowerRegression1a	0 ms	Passed
testPowerRegression1b	0 ms	Passed
testOLSRegression2a	0 ms	Passed
testOLSRegression2b	0 ms	Passed
testPowerRegression2a	0 ms	Passed
testPowerRegression2b	0 ms	Passed

The status bar at the bottom indicates: `Tests passed: 8 of 8 tests - 10 ms`.

נחזיר את הקוד לצורך הלקוח (ביטול משתנים זמינים $res1$ $res2$)

The screenshot shows an IDE interface with two tabs: `Regression.java` and `RegressionTests.java`. The `Regression.java` tab displays the following code:

```
934         result[1] = res1;
935         result[0] = res2;
936         return result;
937     }
938 */
939
940 //result bucketing - extract methods - back to orig code - final version
941 @
942     public static double[] getPowerRegression(XYDataset data, int series) {
943         int n = data.getItemCount(series);
944         if (n < 2) {
945             throw new IllegalArgumentException("Not enough data.");
946         }
947         double sumX = 0;
948         double sumY = 0;
949         double sumXX = 0;
950         double sumXY = 0;
951         for (int i = 0; i < n; i++) {
952             double x = Math.log(data.getXValue(series, i));
953             double y = Math.log(data.getYValue(series, i));
954             sumX += x;
955             sumY += y;
956             double xx = x * x;
957             sumXX += xx;
958             double xy = x * y;
959             sumXY += xy;
960         }
961         double[] result = new double[2];
962         result[1] = getRes1(n, sumX, sumY, sumXX, sumXY);
963         result[0] = getRes2Regression(n, sumX, sumY, result[1]);
964         return result;
965     }
```

The `RegressionTests.java` tab shows the following test results:

Test	Time	Details
Test Results	10 ms	/Library/Java/JavaVirtualMachines/jdk-10
org.jfree.data.statistics.junit.Regr	10 ms	Process finished with exit code 0
testOLSRegression1a	2 ms	
testOLSRegression1b	7 ms	
testPowerRegression1a	0 ms	
testPowerRegression1b	0 ms	
testOLSRegression2a	1 ms	
testOLSRegression2b	0 ms	
testPowerRegression2a	0 ms	
testPowerRegression2b	0 ms	

- כתת נסימ את מה שהיא מטרת ועיקר ביצוע של אלגוריתם ה *bucketing* נבטל את כל ה~~כפלויות 3~~ *Inline Method* ע"י ביצוע של *Type Clone* עברו השיטות החדשות שקיבנו:

השיטה 1 *Getres1* תהיה כתת בשימוש במחלקה 4 פעמים!

ו *getres2regression* פעמיים.

באוטו אופן שביצענו נוכל ליזא גם את *getres2oldregression* והוא תהיה בשימוש פעמיים במחלקה.

```

Regression.java
171 //after inline method - getRes1(n, sumX, sumY, sumXX, sumXY);
172 @ ... public static double[] getOLSRegression(XYDataset data, int series) {
173     int n = data.getItemCount(series);
174     if (n < 2) {
175         throw new IllegalArgumentException("Not enough data.");
176     }
177     double sumX = 0;
178     double sumY = 0;
179     double sumXX = 0;
180     double sumXY = 0;
181     for (int i = 0; i < n; i++) {
182         double x = data.getValue(series, i);
183         double y = data.getYValue(series, i);
184         sumX += x;
185         sumY += y;
186         sumXX += x * x;
187         sumXY += x * y;
188         sumXY += xy;
189     }
190     double Xbar = sumX / n;
191     double Ybar = sumY / n;
192     double[] result = new double[2];
193     result[1] = getRes1(n, sumX, sumY, sumXX, sumXY);
194     result[0] = Ybar - result[1] * Xbar;
195     return result;
196 }
197
198 Regression.getOLSRegression()

RegressionTests.java
99 //after inline method - getRes1(n, sumX, sumY, sumXX, sumXY);
100 @ ... public static double[] getOLSRegression(double[][] data) {
101     int n = data.length;
102     if (n < 2) {
103         throw new IllegalArgumentException("Not enough data.");
104     }
105     double sumX = 0;
106     double sumY = 0;
107     double sumXX = 0;
108     double sumXY = 0;
109     for (int i = 0; i < n; i++) {
110         double x = data[i][0];
111         double y = data[i][1];
112         sumX += x;
113         sumY += y;
114         sumXX += x * x;
115         sumXY += x * y;
116         sumXY += xy;
117     }
118     double Xbar = sumX / n;
119     double Ybar = sumY / n;
120     double[] result = new double[2];
121     result[1] = getRes1(n, sumX, sumY, sumXX, sumXY);
122     result[0] = Ybar - result[1] * Xbar;
123     return result;
124 }

Regression.java
186 //result bucketing - extract methods - back to orig code - final version
187 @ ... public static double[] getPowerRegression(XYDataset data, int series) {
188     int n = data.getItemCount(series);
189     if (n < 2) {
190         throw new IllegalArgumentException("Not enough data.");
191     }
192     double sumX = 0;
193     double sumY = 0;
194     double sumXX = 0;
195     double sumXY = 0;
196     for (int i = 0; i < n; i++) {
197         double x = Math.log(data.getValue(series, i));
198         double y = Math.log(data.getYValue(series, i));
199         sumX += x;
200         sumY += y;
201         double xx = x * x;
202         sumXX += xx;
203         double xy = x * y;
204         sumXY += xy;
205     }
206     double[] result = new double[2];
207     result[1] = getRes1(n, sumX, sumY, sumXX, sumXY);
208     result[0] = getRes2Regression(n, sumX, sumY, result[1]);
209     return result;
210 }

Regression.java
286 //after inline method - getRes1(n, sumX, sumY, sumXX, sumXY); - BUCKETING
287 //getResRegression(n, sumX, sumY, result[])
288 @ ... public static double[] getPowerRegression(double[][] data) {
289     int n = data.length;
290     if (n < 2) {
291         throw new IllegalArgumentException("Not enough data.");
292     }
293     double sumX = 0;
294     double sumY = 0;
295     double sumXX = 0;
296     double sumXY = 0;
297     for (int i = 0; i < n; i++) {
298         double x = Math.log(data[i][0]);
299         double y = Math.log(data[i][1]);
300         sumX += x;
301         sumY += y;
302         double xx = x * x;
303         sumXX += xx;
304         double xy = x * y;
305         sumXY += xy;
306     }
307     double[] result = new double[2];
308     result[1] = getRes1(n, sumX, sumY, sumXX, sumXY);
309     result[0] = getRes2Regression(n, sumX, sumY, result[1]);
310     return result;
311 }
```

נרי'ז טוטים ונבדוק שלא נהרס כלום:

The screenshot shows an IDE interface with several windows open, illustrating the development and testing process for a regression library.

Code Editors:

- Regression.java:** Contains methods for OLS and Power regression calculations, including inline comments and logic for summing values and calculating coefficients.
- RegressionTests.java:** Contains test cases for the regression methods, including assertions for various data points.

Run Tab:

- Shows the command used to run the tests: `/Library/Java/JavaVirtualMachines/jdk-18.0.1.jdk/Contents/Home/bin/java ...`
- Indicates that 8 tests passed in 10 ms.
- Shows the test results for `org.jfree.data.statistics.JUnitReg`, listing individual test cases like `testOLSRegression1a` through `testPowerRegression2b`.

Event Log:

- Logs from 13:46 to 14:00 showing compilation and test completion times.
- Final log entry: "14:00 Tests passed: 8"

סיכום

בדוגמה זאת הראנו כיצד ניתן להשתמש ב- *bucketing* לצורך ביטול כפליות קוד.

לאחר מכן יצאנו 2 שיטות חדשות עי *extract method* והחלפנו את כל כפליות הקוד *Type3* שהוא במחלקה.

סה"כ החלפנו 4 כפליות של מתודה 1, ועוד 2 של מתודה 2.

הכנסנו את המתודות החדשות במקופ כפליות הקוד במחלקה לאורך כל הדרך בקדנו שנוכנות הקוד נשאה ועובדת.

תרונות

- קיבלנו קוד גנרי יותר במחלקה
- חסכנו שימושי קוד רבים בין השיטות השונות שביצענו את אותו הקוד לחישוב המשתנה *res1*
- וב2 שיטות *res2*
- יצרנו אפשרות לשנות בכל רגע נתון את אופן החישוב של המשתנים רק במקום 1 במקום לשנות את אותו השינוי ב6 מקומות
- מניעת שכפולי הקוד ממזערת את הסיכונים לבאים (העתקה לא נcona וכו)
- ומאפשרת בדיקה של חלקיקי קוד בצורה קלה יותר
- ניתן להבין הרבה יותר בקלות מה הקוד עושים

חסרונות

- לדעתי אין-Callo עבור השינוי שביצענו הקוד קרייא יותר קצר מאשר מסודר חוסך כפליות ומאפשר גנריות של הקוד פרט ל-
- זמן העבודה וההשקעה לביצוע האלגוריתם וטעויות בדרך

מקנות

שימוש ב *bucketing* לצורך ביטול כפליות מהסוג זהה הוא טוב כפליות בהן אין לולאה הן החזקה של האלגוריתם

והוא מאפשר לנו לגלוות מהו הסדר הנכון של הקוד מבחינת תלויות ללא צורך בבדיקה בדיקות של *sliding* . *ren1, ren2, ren3* כמו ב

כמו כן עוד חזקה של *bucketing* לעומת *sliding* שביצענו היא שבמידה ויחל להיווצר מצב של הרס ב *dataflow* של הקוד האלגוריתם יגיד לנו היכן למקם את השורות הביעתיות הנ"ל *(before,Marked,after)* וידיע לנו במידה יש שורות שלא קריטיות לכך או لكن נדע גם זאת.

ה *sliding* לעומת זאת חלש בדבר זהה היות ואני רק מקבל רודורדים בהם יכולה להיווצר בעיה ע"י *pen* אך האלגוריתם לא מציע לנו איך לפתרו אותה .

המשך דוגמה נדגים איר Sliding fail

נרצה להציג לסייעם אחרון את חולשתו של *the sliding* לעומת הדוגמא שהראנו כעת ב*bucketing* נבצע את שלבי האלגוריתם הנ"ל בזריזות ולא הרחבה היות והרחבונו והדגמנו רבות שימוש באלגוריתם נדגים שוב גם בדוגמה הבאה:

$V=\{res1\}$

$$\begin{aligned} FlowToExit_{nonv} = & \{ < (Entry, Exit), sumx, sumy, sumxx, sumy > \\ & < (859, Exit), sxx > \\ & < (860, Exit), sxy > \\ & < (861, Exit), xbar > \\ & < (862, Exit), ybar > \\ & < (863, Exit), res2 > \} \end{aligned}$$

чисוב N

$N_v = \{Entry, Exit, 863, 860, 859\}$

$N_{cov} = \{Entry, Exit, 864, 864, 860, 859, 861, 862\}$

$pen1 = \{863, res1\}$

$pen2 = \{\}$

סיכום

קיבלנו כי ב *slices* יכול את אותן השורות כמו ה *bucketing* אבל שהוא *coslice* יכול את כל השורות שעליהן הפעילו את האלגוריתם כמו שטענו אלגוריתם ה *sliding slices* אינו יודע להתמודד ולהציג לנו כיצד לבטל כפליות בהן צריך לסדר מחדש את שורות הקוד ועדין לשמר על נוכנות הקוד, וזהי חזקתו של האלגוריתם של הבאקטיניג וחולשתו של הסlidיניג

RTWQ- Replace Temp with Query

מחלקות קוד: XYErrorRenderer.java

מחלקות טסיטים: XYErrorRendererTest.java

מטרת הריפורן שבסעיפים:

כפי שניתן לראות הקוד ארוך מאוד ומסורבל

ננסה להפוך את הקוד לקריא יותר ומסורבל פחות וה חישוב של המשתנים הזמניים לכן שכל אחד יחוש בפני עצמו ויהי אפשר לשנותו בקלות.

לפנינו שנתחיל לבצע כל דבר על הקוד נרצה לראות קיום של טסיטים ולוזיאו שהם נכשלים בבדיקה כדי שעשינו לדוגמאות הקודמות.

The screenshot shows an IDE interface with two tabs open: XYErrorRenderer.java and XYErrorRendererTests.java. The XYErrorRenderer.java tab displays the Java code for the drawItem method, which handles drawing error bars for a specific data series. The XYErrorRendererTests.java tab shows the test results for the XYErrorRendererTest class, indicating that all 5 tests have passed in 159 ms. The test results are as follows:

Test	Time
Test Results	159 ms
org.jfree.chart.renderer.xy.junit.XYErrorRendererTest	159 ms
testEquals	98 ms
testHashCode	1 ms
testCloning	2 ms
testPublicCloneable	0 ms
testSerialization	60 ms

cut נתבון בפונקציה - הפונקציה הנו"ל מורכבת מ 2 חלקים אשר מחשבת את הערכים של משתנים זמניים.

נתבון בקוד של הפונקציה:

```
284 public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea, PlotRenderingInfo info, XYPlot plot,  
285     ValueAxis domainAxis, ValueAxis rangeAxis, XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass) {  
286     if (pass == 0 && dataset instanceof IntervalXYDataset)  
287         && getitemVisible(series, item)) {  
288         IntervalXYDataset ixyd = (IntervalXYDataset) dataset;  
289         PlotOrientation orientation = plot.getOrientation();  
290         if (this.drawXError) {  
291             // draw the error bar for the x-interval  
292             double x0 = ixyd.getStartXValue(series, item);  
293             double x1 = ixyd.getEndXValue(series, item);  
294             double y = ixyd.getValue(series, item);  
295             RectangleEdge edge = plot.getDomainAxisEdge();  
296             double xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);  
297             double xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);  
298             double yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());  
299             Line2D line;  
300             Line2D cap1 = null;  
301             Line2D cap2 = null;  
302             double adj = this.capLength / 2.0;  
303             if (orientation == PlotOrientation.VERTICAL) {  
304                 line = new Line2D.Double(xx0, yy, xx1, yy);  
305                 cap1 = new Line2D.Double(xx0, yy - adj, xx0, yy + adj);  
306                 cap2 = new Line2D.Double(xx1, yy - adj, xx1, yy + adj);  
307             }  
308             else if (...) PlotOrientation.HORIZONTAL  
309                 line = new Line2D.Double(yy, xx0, yy, xx1);  
310                 cap1 = new Line2D.Double(xx0, yy - adj, xx0, yy + adj);  
311                 cap2 = new Line2D.Double(xx1, yy - adj, xx1, yy + adj);  
312             }  
313             g2.setStroke(new BasicStroke( width: 1.0f));  
314             if (this.errorPaint != null) {  
315                 g2.setPaint(this.errorPaint);  
316             }  
317             else {  
318                 g2.setPaint(getItemPaint(series, item));  
319             }  
320             g2.draw(line);  
321             g2.draw(cap1);  
322             g2.draw(cap2);  
323         }  
324     }  
325     super.drawItem(g2, state, dataArea, info, plot, domainAxis, rangeAxis,  
326                     dataset, series, item, crosshairState, pass);  
327 }
```

```
304     if (this.drawYError) {  
305         // draw the error bar for the y-interval  
306         double y0 = ixyd.getStartYValue(series, item);  
307         double y1 = ixyd.getEndYValue(series, item);  
308         double x = ixyd.getValue(series, item);  
309         RectangleEdge edge = plot.getRangeAxisEdge();  
310         double yy0 = rangeAxis.valueToJava2D(y0, dataArea, edge);  
311         double yy1 = rangeAxis.valueToJava2D(y1, dataArea, edge);  
312         double xx = domainAxis.valueToJava2D(x, dataArea,  
313             plot.getDomainAxisEdge());  
314         Line2D line;  
315         Line2D cap1 = null;  
316         Line2D cap2 = null;  
317         double adj = this.capLength / 2.0;  
318         if (orientation == PlotOrientation.VERTICAL) {  
319             line = new Line2D.Double(xx, yy0, xx, yy1);  
320             cap1 = new Line2D.Double(xx - adj, yy0, xx + adj, yy0);  
321             cap2 = new Line2D.Double(xx - adj, yy1, xx + adj, yy1);  
322         }  
323         else { // PlotOrientation.HORIZONTAL  
324             line = new Line2D.Double(yy0, xx, yy1, xx);  
325             cap1 = new Line2D.Double(yy0, xx - adj, yy0, xx + adj);  
326             cap2 = new Line2D.Double(yy1, xx - adj, yy1, xx + adj);  
327         }  
328         g2.setStroke(new BasicStroke( width: 1.0f));  
329         if (this.errorPaint != null) {  
330             g2.setPaint(this.errorPaint);  
331         }  
332         else {  
333             g2.setPaint(getItemPaint(series, item));  
334         }  
335         g2.draw(line);  
336         g2.draw(cap1);  
337         g2.draw(cap2);  
338     }  
339 }  
340 super.drawItem(g2, state, dataArea, info, plot, domainAxis, rangeAxis,  
341                 dataset, series, item, crosshairState, pass);  
342 }
```

```
272     double x0 = ixyd.getStartXValue(series, item);  
273     double x1 = ixyd.getEndXValue(series, item);  
274     double y = ixyd.getYValue(series, item);  
275     RectangleEdge edge = plot.getDomainAxisEdge();  
276     double xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);  
277     double xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);  
278     double yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());  
279     Line2D line;  
280     Line2D cap1 = null;  
281     Line2D cap2 = null;  
282     double adj = this.capLength / 2.0;  
283     if (orientation == PlotOrientation.VERTICAL) {  
284         line = new Line2D.Double(xx0, yy, xx1, yy);  
285         cap1 = new Line2D.Double(xx0, yy - adj, xx0, yy + adj);  
286         cap2 = new Line2D.Double(xx1, yy - adj, xx1, yy + adj);  
287     }  
288     else { // PlotOrientation.HORIZONTAL  
289         line = new Line2D.Double(yy, xx0, yy, xx1);  
290         cap1 = new Line2D.Double( yy, yy - adj, xx0, yy + adj);  
291         cap2 = new Line2D.Double( yy, yy - adj, xx1, yy + adj);  
292     }  
293     g2.setStroke(new BasicStroke(width: 1.0f));
```

נרצה לבצע $RTWQ$ כאשר $V = \{line\}$

להלן האלגוריתם של $RTWQ$

Mechanics for Replace Temp with Query

- Identify the relevant fragment of code, from the temp's declaration to the end of its enclosing block.
- Perform sliding on that code fragment for the selected temp.
 - *If the sliding fails you may want to choose a different temp to replace with a query.*
 - *Successful sliding will bring together the code for computing the final value of the temp, making it a contiguous fragment ready to be extracted into a method of its own.*
- Compile and test.
- Perform Extract Method on the extracted slice, giving it an appropriate name.
 - *If the extracted method appears to have side effects consider extracting a different temp, or modify the code to prevent the side effects.*
 - *If all those side effects occur in invoked methods, consider applying Separate Query from Modifier on those methods before re-applying this refactoring. That way, the extracted modifiers could possibly be excluded from the query, and cause no side effects.*
- Compile and test.
- Perform Inline Temp on the selected temp at its declaration.
 - *This step involves the replacement of all references to the temp with the call to the extracted method (i.e. the query), and the removal of the temp's declaration.*
 - *If the value of any of the query's parameters may be different at any point of reference, consider adding backup variables at the temp's point of declaration, or abandon the refactoring.*
 - *A potential cause of failure is the need of backup for a non-cloneable parameter object; making such backup might otherwise not be desirable due to space (i.e. large object to clone) or other considerations.*
- Compile and test.

שלב 1

מציאת הסkop של המשתנה הזמן שנרצה לבדוק $line =$ נבחן כי $line$ מוגדר פעמיים ב-2 סkopים שונים בפונקציה אנו מתמקדים ב $line$ הראשון ולכן הסkop שייעני אותנו בנקודתה בה הערך של $line$ קבוע הוא בשורות 291-272.

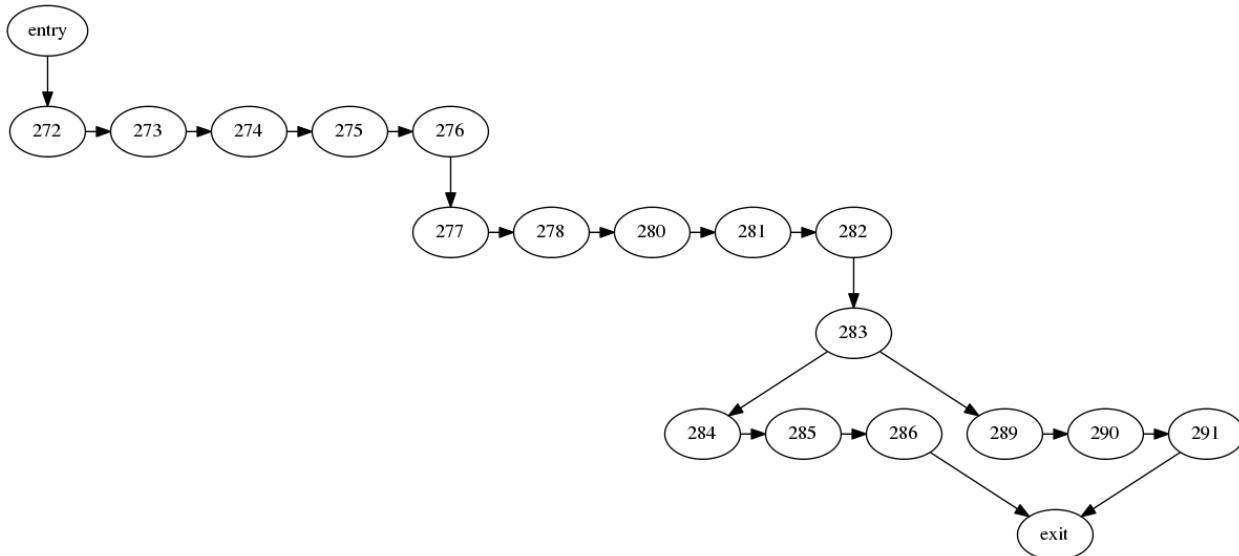
שלב 2

ביצוע *sliding* על קטע קוד הנ"ל עבור $\{line\} =$ (האלגוריתם נמצא בדוגמה הראשונה)

שלבי ביצוע *sliding*

בנייה CFG

שורה 279-272- הצהרה ולכן לא נכללת בגרפים



$E_c = \{(Entry, 272), (Entry, 273), (Entry, 274), (Entry, 275), (Entry, 276),$
 $(Entry, 277), (Entry, 278),$
 $(Entry, 280), (Entry, 281), (Entry, 282), (Entry, 283), (283,284),$
 $(283,285), (283,286), (283,289), (283,290), (283,291)\}$

קשרות DATA

משתנה	זיהוי	קשר	#
{ series, item, ixyb.[x] }	flow	(Entry,272)	1
{ series, item, ixyb.[x] }	flow	(Entry,273)	2
{ series, item, ixyb.[x] }	flow	(Entry,274)	3
{ plot.[x] }	flow	(Entry,275)	4
{ domainAxi.[x],dataArea }	flow	(Entry,276)	5
{x0}	flow	(272,276)	6
{edge}	flow	(275,276)	7
{ domainAxi.[x],dataArea }	flow	(Entry,277)	8
{x1}	flow	(273,277)	9
{edge}	flow	(275,277)	10
{ rangetaxis.[x], dataArea, plot.[x] }	flow	(Entry,278)	11
{ y }	flow	(274,278)	12
{ this.[x] }	flow	(Entry,282)	13
{orientation,this[x]}	flow	(Entry,283)	14
{xx0}	flow	(276,284)	15
{yy}	flow	(278,284)	16
{xx1}	flow	(277,284)	17
{xx0}	flow	(276,285)	18
{yy}	flow	(278,285)	19
{adj}	flow	(282,285)	20
{xx1}	flow	(277,286)	21
{yy}	flow	(278,286)	22
{adj}	flow	(282,286)	23
{yy}	flow	(278,289)	24
{xx0}	flow	(276,289)	25
{xx1}	flow	(277,289)	26
{yy}	flow	(278,290)	27
{adj}	flow	(282,290)	28
{xx0}	flow	(276,290)	29

{yy}	flow	(278,291)	30
{adj}	flow	(282,291)	31
{xx1}	flow	(277,291)	32
{line}	flow	(284, Exit)	33
{line}	flow	(289, Exit)	34
{cap1}	flow	(285, Exit)	35
{cap1}	flow	(290, Exit)	36
{cap2}	flow	(286, Exit)	37
{cap2}	flow	(291, Exit)	38

שלב 4-1 באלגוריתם הולידייניג

$FlowToExit_{nonv} = \{ < (Entry, Exit) >$
 $< series, item, g2 >$
 $< (285, exit), cap1 >$
 $< (290, exit), cap1 >$
 $< (286, exit), cap2 >$
 $< (291, exit), cap2 > \}$

שלב 5 חישוב N_v

$$N_v = \{289, Exit, 284, 283, 282, 278, 277, 276, 275, 274, 273, 272\}$$

שלב 6-9

$$\text{Non Final Uses At Node} = \left\{ \begin{array}{l} (285, cap1), (290, cap1), (284, line), (289, line), \\ (286, cap2), (291, cap2) \end{array} \right\}$$

שלב 10-13

$$Flow To Final Uses_v = \left\{ \begin{array}{l} (284, exit), \{line\} \\ (289, exit), \{line\} \end{array} \right\}$$

שלב 14 – חישוב ה coslice

$$N_{cov} = \{Entry, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281\}$$
$$\quad \quad \quad 282, 283, 285, 286, 288, 290, 291, Exit\}$$

שלב 15-16 פיצויים

Pen1 = {}, Pen2 = {}, Pen3 = {},

שלב 17

$$N_v = \{289, Exit, 284, 283, 282, 278, 277, 276, 275, 274, 273, 272\}$$

$$N_{cov} = \{Entry, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281\}$$
$$\quad \quad \quad 282, 283, 285, 286, 288, 290, 291, Exit\}$$

Pen1 = {}, Pen2 = {}, Pen3 = {}

הקוד שקיבלו לאחר ביצוע סליידינג על `:v={line}`

Slice:

```
//slice
double x0 = ixyd.getStartXValue(series, item);
double x1 = ixyd.getEndXValue(series, item);
double y = ixyd.getYValue(series, item);
RectangleEdge edge = plot.getDomainAxisEdge();
double xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
double xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
double yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
Line2D line;
double adj = this.capLength / 2.0;
if (orientation == PlotOrientation.VERTICAL) {
    line = new Line2D.Double(xx0, yy, xx1, yy);
}
else { // PlotOrientation.HORIZONTAL
    line = new Line2D.Double(yy, xx0, yy, xx1);
}
```

Co-Slice:

```
//co - slice
// draw the error bar for the x-interval
x0 = ixyd.getStartXValue(series, item);
x1 = ixyd.getEndXValue(series, item);
y = ixyd.getYValue(series, item);
edge = plot.getDomainAxisEdge();
xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());

Line2D cap1 = null;
Line2D cap2 = null;
adj = this.capLength / 2.0;
if (orientation == PlotOrientation.VERTICAL) {
    cap1 = new Line2D.Double(xx0, y - adj, xx0, y + adj);
    cap2 = new Line2D.Double(xx1, y - adj, xx1, y + adj);
}
else { // PlotOrientation.HORIZONTAL
    cap1 = new Line2D.Double(x1 - adj, yy, x1 + adj, yy);
    cap2 = new Line2D.Double(x1 - adj, yy, x1 + adj, yy);
}
```

שלב הבא באlgorigthm RTWQ

נקمل ונרייך טסיטים כדי לבדוק שלא הרסנו כלום בנסיבות הפונקציה

- נשים לב שכך שיתקמל אנו חייבים להוריד את ההצלחות של המשתנים שהועברו לו *coslice* לאחר *extract method* נctrar להחזיר

The screenshot shows an IDE interface with two code files open:

- XYErrorRenderer.java**: Contains Java code for rendering error bars. It includes logic for both vertical and horizontal slices, involving `IntervalXYDataset`, `PlotOrientation`, and `Line2D`.
- XYErrorRendererTests.java**: Contains JUnit test cases for the `XYErrorRenderer`. The tests are grouped under `Test Results` and include:
 - `testEquals`: 100 ms
 - `testHashCode`: 1 ms
 - `testCloning`: 3 ms
 - `testPublicCloneable`: 0 ms
 - `testSerialization`: 75 ms

The bottom right corner of the IDE shows the test results: "Tests passed: 5 of 5 tests – 179 ms".

שלב 4 Extract Metod

نبצע על ה *slice* שקבענו

```
private Line2D getLine2D(Rectangle2D dataArea, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
                        int series, int item, IntervalXYDataset ixyd, PlotOrientation orientation) {
    double x0 = ixyd.getStartXValue(series, item);
    double x1 = ixyd.getEndXValue(series, item);
    double y = ixyd.getYValue(series, item);
    RectangleEdge edge = plot.getDomainAxisEdge();
    double xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
    double xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
    double yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
    Line2D line;
    double adj = this.capLength / 2.0;
    if (orientation == PlotOrientation.VERTICAL) {
        line = new Line2D.Double(xx0, yy, xx1, yy);
    }
    else { // PlotOrientation.HORIZONTAL
        line = new Line2D.Double(yy, xx0, yy, xx1);
    }
    return line;
}
```

להלן הקוד של הפונקציה לאחר פעולה זאת.

```
//after sliding v={line} lines 272-291 - extract method getLine2D
public void drawItem(Graphics2D g2, XYItemRendererState state, Rectangle2D dataArea, PlotRenderingInfo info, XYPlot plot,
                      ValueAxis domainAxis, ValueAxis rangeAxis, XYDataset dataset, int series, int item, CrosshairState crosshairState, int pass) {
    if (pass == 0 && dataset instanceof IntervalXYDataset
        && getItemVisible(series, item)) {
        IntervalXYDataset ixyd = (IntervalXYDataset) dataset;
        PlotOrientation orientation = plot.getOrientation();
        if (this.drawError) {
            // draw the error bar for the x-interval
            //slice
            Line2D line = getLine2D(dataArea, plot, domainAxis, rangeAxis, series, item, ixyd, orientation);
            double x0;
            double x1;
            double y;
            RectangleEdge edge;
            double xx0;
            double xx1;
            double yy;
            double adj;
            //co - slice
            // draw the error bar for the x-interval
            x0 = ixyd.getStartXValue(series, item);
            x1 = ixyd.getEndXValue(series, item);
            y = ixyd.getYValue(series, item);
            edge = plot.getDomainAxisEdge();
            xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
            xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
            yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
            Line2D cap1 = null;
            Line2D cap2 = null;
            adj = this.capLength / 2.0;
            if (orientation == PlotOrientation.VERTICAL) {
                cap1 = new Line2D.Double(xx0, y - adj, xx0, y + adj);
                cap2 = new Line2D.Double(xx1, y - adj, xx1, y + adj);
            }
            else { // PlotOrientation.HORIZONTAL
                cap1 = new Line2D.Double(x0: y - adj, xx0: y + adj, xx0: y + adj, xx1);
                cap2 = new Line2D.Double(x1: y - adj, xx1: y + adj, xx1: y + adj, xx1);
            }
        }
    }
}
```

שלב 5 נקמפל ונרץ טסטים

The screenshot shows an IDE interface with two tabs open: XYErrorRendererTests.java and XYErrorRenderer.java. The XYErrorRendererTests.java tab displays a portion of the code, specifically the drawItem method, which handles drawing error bars for data points. The code uses Java 2D graphics and various classes like Graphics2D, XYItemRendererState, Rectangle2D, PlotRenderingInfo, XYPlot, ValueAxis, IntervalXYDataset, PlotOrientation, and Line2D. The XYErrorRenderer.java tab is partially visible at the top.

Below the code editor is a 'Run' panel titled 'XYErrorRendererTests'. It shows a green play button icon followed by the text 'Tests passed: 5 of 5 tests - 196 ms'. Underneath, there's a tree view of test results:

- Test Results (196 ms)
- org.jfree.chart.renderer.xy.junit.XYErrorRendererTests (196 ms)
 - testEquals (115 ms)
 - testHashCode (2 ms)
 - testCloning (3 ms)
 - testPublicCloneable (1 ms)
 - testSerialization (75 ms)

At the bottom right of the run panel, there is a small green box containing the text 'Tests passed: 5'.

שלב 6 inline temp RTWQ

מחליף את כל הקריאה ל *line* בפונקציה החדשה

The screenshot shows an IDE interface with two main windows. The top window is a code editor displaying Java code for an *XYErrorRenderer* class. The bottom window is a 'Run' window titled 'XYErrorRendererTests' showing the execution results.

Code Editor Content:

```
463     double adj;
464     //co - slice
465     // draw the error bar for the x-interval
466     x0 = ixyd.getStartXValue(series, item);
467     x1 = ixyd.getEndXValue(series, item);
468     y = ixyd.getYValue(series, item);
469     edge = plot.getDomainAxisEdge();
470     xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
471     xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
472     yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
473     Line2D cap1 = null;
474     Line2D cap2 = null;
475     adj = this.capLength / 2.0;
476     if (orientation == PlotOrientation.VERTICAL) {
477         cap1 = new Line2D.Double(xx0, y: yy - adj, xx0, y2: yy + adj);
478         cap2 = new Line2D.Double(xx1, y: yy - adj, xx1, y2: yy + adj);
479     } else { // PlotOrientation.HORIZONTAL
480         cap1 = new Line2D.Double( x1: yy - adj, xx0, x2: yy + adj, xx0);
481         cap2 = new Line2D.Double( x1: yy - adj, xx1, x2: yy + adj, xx1);
482     }
483     g2.setStroke(new BasicStroke( width: 1.0f));
484     if (this.errorPaint != null) {
485         g2.setPaint(this.errorPaint);
486     } else {
487         g2.setPaint(getItemPaint(series, item));
488     }
489     g2.draw(getLine2D(dataArea, plot, domainAxis, rangeAxis, series, item, ixyd, orientation));
490     g2.draw(cap1);
491     g2.draw(cap2);
492 }
493 if (this.drawYError) {
494     // draw the error bar for the y-interval
495 }
```

Run Window Content:

Tests passed: 5 of 5 tests – 298 ms

Test	Time
Test Results	298 ms
org.jfree.chart.renderer.xy.junit	298 ms
testEquals	165 ms
testHashCode	2 ms
testCloning	3 ms
testPublicCloneable	0 ms
testSerialization	128 ms

Process finished with exit code 0

שלב 7 נקמפל ונריצ'

The screenshot shows an IDE interface with the following details:

- Code Editor:** Displays the `XYErrorRendererTests.java` file, specifically the `drawItem` method implementation.
- Run Results:** Shows the output of the test run:
 - Tests passed: 5 of 5 tests – 172 ms
 - Process finished with exit code 0
 - Test Results section:
 - org.jfree.chart.renderer.xy.junit.XYErrorRendererTest (172 ms)
 - testEquals (105 ms)
 - testHashCode (1 ms)
 - testCloning (2 ms)
 - testPublicCloneable (0 ms)
 - testSerialization (64 ms)
 - A green button at the bottom right says "Tests passed: 5"

מבצע שוב RTW - עבור {cap1} V=

כעת נציג את הקוד לפני ואחרי ללא חזרה על כל השלבים כיון שהציגנו בהרחבה את השלבים בדוגמאות האחרות של ביצוע הסליידינג.

קוד לפניה ביצוע שלב 1 – של סליידינג

```
466     x0 = ixyd.getStartXValue(series, item);
467     x1 = ixyd.getEndXValue(series, item);
468     y = ixyd.getYValue(series, item);
469     edge = plot.getDomainAxisEdge();
470     xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
471     xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
472     yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
473     Line2D cap1 = null;
474     Line2D cap2 = null;
475     adj = this.capLength / 2.0;
476     if (orientation == PlotOrientation.VERTICAL) {
477         cap1 = new Line2D.Double(xx0, y1: yy - adj, xx0, y2: yy + adj);
478         cap2 = new Line2D.Double(xx1, y1: yy - adj, xx1, y2: yy + adj);
479     }
480     else { // PlotOrientation.HORIZONTAL
481         cap1 = new Line2D.Double( x1: yy - adj, xx0, x2: yy + adj, xx0);
482         cap2 = new Line2D.Double( x1: yy - adj, xx1, x2: yy + adj, xx1);
483     }
484     g2.setStroke(new BasicStroke( width: 1.0f));
485     if (this.errorPaint != null) {
486         g2.setPaint(this.errorPaint);
487     }
488     else {
489         g2.setPaint(getItemPaint(series, item));
490     }
491     g2.draw(getLine2D(dataArea, plot, domainAxis, rangeAxis, series, item, ixyd, orientation));
492     g2.draw(cap1);
493     g2.draw(cap2);
494 }
```

קוד אחריו ביצוע שלב 1 של סליידינג

```
double adj,  
//slice  
x0 = ixyd.getStartXValue(series, item);  
y = ixyd.getYValue(series, item);  
edge = plot.getDomainAxisEdge();  
xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);  
yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());  
Line2D cap1 = null;  
adj = this.capLength / 2.0;  
if (orientation == PlotOrientation.VERTICAL) {  
    cap1 = new Line2D.Double(xx0, y1: yy - adj, xx0, y2: yy + adj);  
}  
else { // PlotOrientation.HORIZONTAL  
    cap1 = new Line2D.Double(x1: yy - adj, xx0, x2: yy + adj, xx0);  
}  
//coslice  
x1 = ixyd.getEndXValue(series, item);  
y = ixyd.getYValue(series, item);  
edge = plot.getDomainAxisEdge();  
xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);  
yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());  
Line2D cap2 = null;  
adj = this.capLength / 2.0;  
if (orientation == PlotOrientation.VERTICAL) {  
    cap2 = new Line2D.Double(xx1, y1: yy - adj, xx1, y2: yy + adj);  
}  
else { // PlotOrientation.HORIZONTAL  
    cap2 = new Line2D.Double(x1: yy - adj, xx1, x2: yy + adj, xx1);  
}
```

The screenshot shows an IDE interface with two main panes. The top pane displays a portion of a Java source file named XYErrorRenderer.java. The code is annotated with line numbers from 555 to 584. It contains logic for rendering error bars, specifically handling vertical and horizontal slices. The bottom pane shows the 'Run' tool window for the 'XYErrorRendererTests' test suite. The test results table lists five tests that have passed, each with a green checkmark, a duration of 187 ms, and the path /Library/Java/JavaVirtualMachines/jdk-10.0.1.j. The message 'Process finished with exit code 0' is also visible.

```

555     double adj;
556     //slice
557     x0 = ixyd.getStartXValue(series, item);
558     y = ixyd.getYValue(series, item);
559     edge = plot.getDomainAxisEdge();
560     xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
561     yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
562     Line2D cap1 = null;
563     adj = this.capLength / 2.0;
564     if (orientation == PlotOrientation.VERTICAL) {
565         cap1 = new Line2D.Double(xx0, y1: yy - adj, xx0, y2: yy + adj);
566     }
567     else { // PlotOrientation.HORIZONTAL
568         cap1 = new Line2D.Double(x1: yy - adj, xx0, x2: yy + adj, xx0);
569     }
570     //coslice
571     x1 = ixyd.getEndXValue(series, item);
572     y = ixyd.getYValue(series, item);
573     edge = plot.getDomainAxisEdge();
574     xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
575     yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
576     Line2D cap2 = null;
577     adj = this.capLength / 2.0;
578     if (orientation == PlotOrientation.VERTICAL) {
579         cap2 = new Line2D.Double(xx1, y1: yy - adj, xx1, y2: yy + adj);
580     }
581     else { // PlotOrientation.HORIZONTAL
582         cap2 = new Line2D.Double(x1: yy - adj, xx1, x2: yy + adj, xx1);
583     }
584     q2.setStroke(new BasicStroke(width: 1.0f));

```

Run: XYErrorRendererTests

	Test Results	Time	Path
1	Test Results	187 ms	/Library/Java/JavaVirtualMachines/jdk-10.0.1.j
2	org.jfree.chart.renderer.xy.junit.XYErrorRendererTe	187 ms	
3	testEquals	107 ms	
4	testHashCode	1 ms	
5	testCloning	3 ms	
6	testPublicCloneable	1 ms	
7	testSerialization	75 ms	

Tests passed: 5 of 5 tests – 187 ms

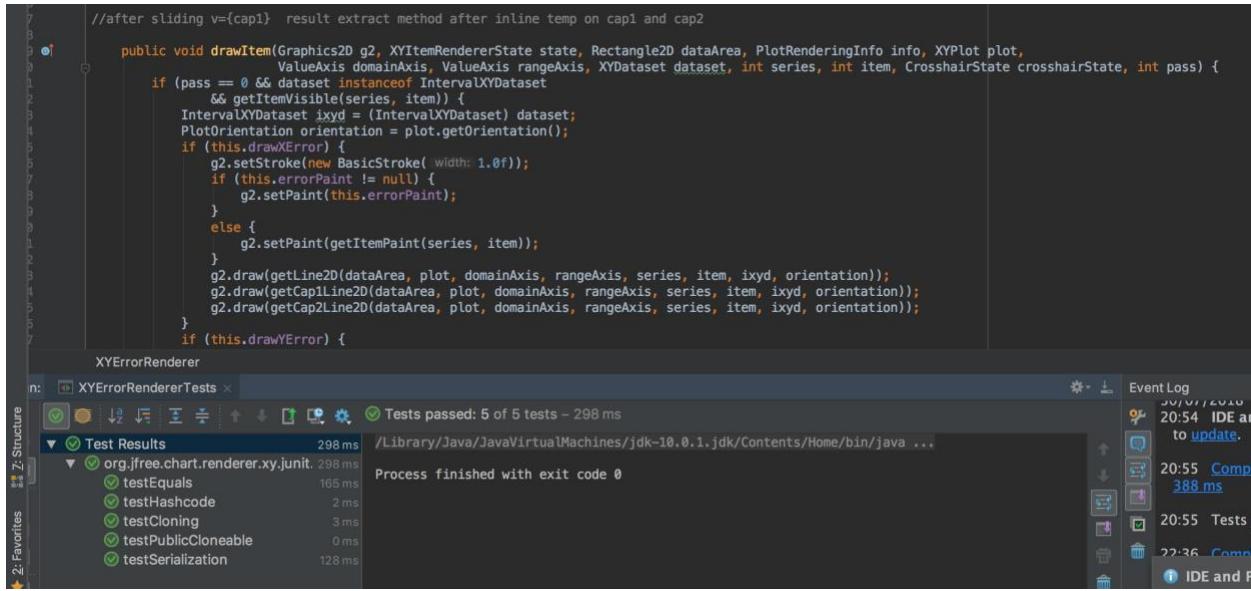
Process finished with exit code 0

בבצע coslice על ה slice ועל ה Extract Method

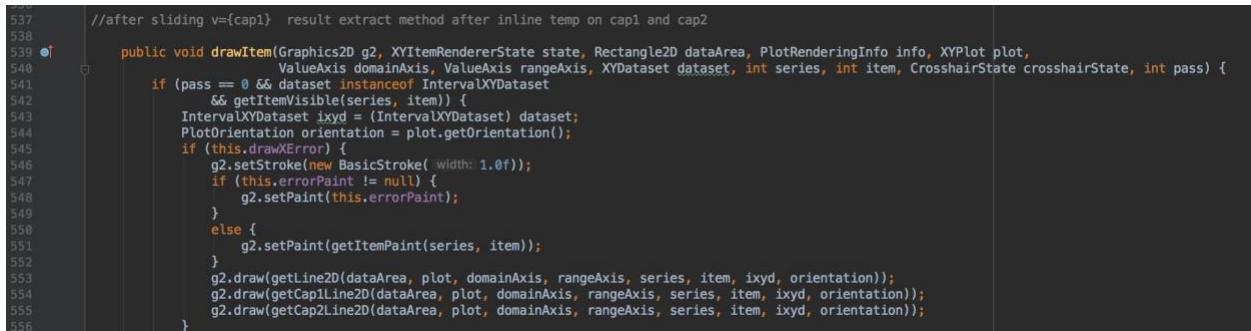
```
private Line2D getCap1Line2D(Rectangle2D dataArea, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis,
                           int series, int item, IntervalXYDataset ixyd, PlotOrientation orientation) {
    double x0;
    double y;
    RectangleEdge edge;
    double xx0;
    double yy;
    double adj;
    x0 = ixyd.getStartXValue(series, item);
    y = ixyd.getYValue(series, item);
    edge = plot.getDomainAxisEdge();
    xx0 = domainAxis.valueToJava2D(x0, dataArea, edge);
    yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
    Line2D cap1 = null;
    adj = this.capLength / 2.0;
    if (orientation == PlotOrientation.VERTICAL) {
        cap1 = new Line2D.Double(xx0, y1: yy - adj, xx0, y2: yy + adj);
    }
    else { // PlotOrientation.HORIZONTAL
        cap1 = new Line2D.Double( x1: yy - adj, xx0, x2: yy + adj, xx0);
    }
    return cap1;
}
```

```
private Line2D getCap2Line2D(Rectangle2D dataArea, XYPlot plot, ValueAxis domainAxis, ValueAxis rangeAxis, int series,
                           int item, IntervalXYDataset ixyd, PlotOrientation orientation) {
    double x1;
    double y;
    RectangleEdge edge;
    double xx1;
    double yy;
    double adj;
    x1 = ixyd.getEndXValue(series, item);
    y = ixyd.getYValue(series, item);
    edge = plot.getDomainAxisEdge();
    xx1 = domainAxis.valueToJava2D(x1, dataArea, edge);
    yy = rangeAxis.valueToJava2D(y, dataArea, plot.getRangeAxisEdge());
    Line2D cap2 = null;
    adj = this.capLength / 2.0;
    if (orientation == PlotOrientation.VERTICAL) {
        cap2 = new Line2D.Double(xx1, y1: yy - adj, xx1, y2: yy + adj);
    }
    else { // PlotOrientation.HORIZONTAL
        cap2 = new Line2D.Double( x1: yy - adj, xx1, x2: yy + adj, xx1);
    }
    return cap2;
}
```

בקטffffff מפל וניהז טווטים



Inline temp



בדוגמא זאת ראיינו כיצד ע"י שימוש ב**NG/D/TS** נוכל לבצע את **QRTW+M** מטרתנו ביצוע זה הייתה לגרום לקוד להיות קרייא יותר. גנרי יותר, ויתר מובן ופחות מסורבל להבנת הקורא. אנו מאמינים כי עמדנו במשימה. הקוד קרייא מאד ונימן להבין בקלות יותר מה הפונקציה עשו.

יתרונות

- הקוד קרייא הרבה יותר
- הורדנו **BED SMELL LONG METHOD**
- ניתן להשתמש ולמחזר את הקוד במידה יש **שייפול** קוד של המשתנים הזמןניים שחילצנו.
- נוכל לשנות את אופן החישוב של כל המשתנים הזמןניים בכל רגע נתון ללא ניסוי להבין האם פוגע באחד מהчисובים האחרים – כמובן הגענו לצימוד נמור יותר בין המשתנים שחילצנו.

חסרונות

קיבלנו 3 פונקציות חדשות ובהן אנו מחשבים כל פעם מחדש את **XXZX0X0X1XXYY1** אומנם השגנו קוד קרייא גנרי יותר אך זה עולה לנו ב trade off של יעילות וביצועים ובהוספה של 3 שיטות נוספות שיתכן ולא יישמו למניעת **שייפול** קוד בעtid.

מסקנות

ביצוע **RTWQ** ע"י סליידינג במקרה זה לדעתנו שווה את זה.

קיבלנו קוד קרייא מסודר ואפשרויות לשינוי אופן חישוב המשתנים בקלות. החזקה של האלגוריתם שהצלחנו לבדוק כל משתנה ולייצא שיטות אך החולשה העיקרית היא שאנחנו מחשבים לכל אחד מהמשתנים 7 משתנים כל פעם למחרת שערכם אינו משתנה.