**אנליזה נומרית תשע"ט**

**דו"ח סיכום האקטון**

קבוצה מס': 23

חברי הקבוצה הנוכחים: שלי מירון, אור ממן, סתיו לובל, איוון רובינסון

**סעיף א – הגדרת הבעיה –**

במאמר שניתן לנו ששמו:  
*“Two independent approaches used for estimating 2d contamination distribution on ground level- based on air monitoring information”*   
מוצגת בעיה מורכבת שדורשת פתרונות נומרים בכדי להביא אותה לידי מימוש תוכנתי.

הבעיה מתארת איזור מזוהם באיזוטופים רדיואקטיביים שיש לכמת, באמצעות הליקופטר עם גלאי/חיישן, שמטרתו למפות את רמות הקרינה באיזור.

הגלאי סורק את שטח ה- NXN המתואר, מגובה מסוים, לפי השיטה הבאה:

השטח מחולק לריבועים אחידים. מרכז כל "תא" כזה (=ריבוע) ייצג את רמת הקרינה בכל התא.

ההליקופטר מושפע מהקרינה בכל השטח בו-זמנית, אבל קיימות נוסחאות מתמטיות המאפשרות לבודד מתוך המדדים את רמות הקרינה בכל יחידת שטח. זאת מטרתינו – לפתור משוואות מתמטיות אלה באופן נומרי.

כדי לקבל תוצאה אמינה, יש לבצע את החישובים בשתי דרכים שונות. אנחנו נתמקד בדרך אחת – פתרון מטריצות.

הקבועים שאנחנו צריכים לקבל על מנת לבצע את החישובים:

* c = proportion coefficient of the monitor [cps m^2/gamma]
* k = radiation build up factor in air [m^-1]
* µ = radiation absorption coefficient in air [m^-1]
* גודל השטח המזוהם [m^2]
* רזולוציית חלוקת השטח לריבועים
* גובה ריחוף המסוק [m]
* המדידות שהתקבלו מהמסוק [cps]

יש לשים לב כי רמת דיוק הקבועים עלולה להשפיע על רמת דיוק התוצאה שנחשב; לכן נשתמש במספר שיטות שונות כדי להגיע אליהם.

כפל מטריצות מטבעו פעולה לא יציבה, לכן יש לשים לב לרמת הדיוק במטריצה שנחשב, ושנשתמש בשיטות הממזערות את רמת השגיאה.

**סעיף ב' – השיטה והצגת הכלים לפתרון –**

1. נחשב את הקבוע c  
   נשתמש ב Bisection Algorithm ו- Secant Algorithm (ראה נספח א‘).  
   ידוע שהקבוע שווה למחצית השורש האמתי  
   של משוואת ליאנרדו:   
   מצאנו שהשורש האמתי הוא:

|  |  |
| --- | --- |
| Secant | |
| Iteration | Result |
| 1 | 5.00000000000000 |
| 2 | 0.44444444444444 |
| 3 | 0.75645284947611 |
| 4 | 1.56030052516560 |
| 5 | 1.33338424878932 |
| 6 | 1.36688417586263 |
| 7 | 1.36882792756969 |
| 8 | 1.36880809678120 |
| 9 | 1.36880810782130 |
| 10 | 1.36880817821372 |
| 11 | **1.36880810782137** |

|  |  |
| --- | --- |
| Bisection | |
| Iteration | Result |
| 1 | 2.50000000000000 |
| 2 | 1.25000000000000 |
| 3 | 1.87500000000000 |
| 4 | 1.56250000000000 |
| 5 | 1.40625000000000 |
| 6 | 1.32812500000000 |
| 7 | 1.36718750000000 |
| 8 | 1.38671875000000 |
| 9 | 1.37695312500000 |
| 10 | 1.37207031250000 |
| 11 | 1.36962890625000 |
| 12 | 1.36840820312500 |
| 13 | 1.36901855468750 |
| 14 | 1.36871337890625 |
| 15 | 1.36886596679687 |
| 16 | 1.36878967285156 |
| 17 | 1.36882781982421 |
| 18 | 1.36880874633789 |
| 19 | 1.36879920959472 |
| 20 | 1.36880397796630 |
| 21 | 1.36880636215209 |
| 22 | 1.36880755424499 |
| … | |
| 54 | **1.36880810782137** |

לפי הנתונים הבאים למטודות:

Secant: findRoots(lambda x:x\*\*3+2\*x\*\*2+10\*x-20, 0, 5, 10)

Bisection: findRoots(lambda x:x\*\*3+2\*x\*\*2+10\*x-20, 0, 5, 0)

כפי שרואים מהטבלאות, הקבוע מתכנס ל- 1.36880810782137 (אבל בשיטת החציה לוקח הרבה יותר איטרציות).

נחלק ב- 2 כדי לקבל את :

**c =** 1.36880810782137/2 = **0.6844040539**

1. נחשב את הקבוע µ  
   נשתמש ב Bisection Algorithm ו- Secant Algorithm (ראה נספח א‘).  
   ידוע שהקבוע שווה לאלפית השורש החיובי של המשוואה:

מצאנו שהשורש החיובי הוא:

|  |  |
| --- | --- |
| Secant | |
| Iteration | Result |
| 1 | 5.000000000000000 |
| 2 | 2.177289211547380 |
| 3 | 2.370671615675610 |
| 4 | 0.796623577194411 |
| 5 | 0.873067689228107 |
| 6 | 0.912823852632486 |
| 7 | 0.909944342286736 |
| 8 | 0.910007467534414 |
| 9 | 0.910007572492629 |
| 10 | 0.910007572488709 |
| **11** | **0.910007572488709** |

|  |  |
| --- | --- |
| Bisection | |
| Iteration | Result |
| 1 | 2.500000000000000 |
| 2 | 1.250000000000000 |
| 3 | 0.625000000000000 |
| 4 | 0.937500000000000 |
| 5 | 0.781250000000000 |
| 6 | 0.859375000000000 |
| 7 | 0.898437500000000 |
| 8 | 0.917968750000000 |
| 9 | 0.908203125000000 |
| 10 | 0.913085937500000 |
| 11 | 0.910644531250000 |
| 12 | 0.909423828125000 |
| 13 | 0.910034179687500 |
| 14 | 0.909729003906250 |
| 15 | 0.909881591796875 |
| 16 | 0.909957885742187 |
| 17 | 0.909996032714843 |
| 18 | 0.910015106201171 |
| 19 | 0.910005569458007 |
| 20 | 0.910010337829589 |
| 21 | 0.910007953643798 |
| 22 | 0.910006761550903 |
| **23** | **0.910007357597351** |

לפי הנתונים הבאים למטודות:

Secant: findRoots(lambda x:math.exp(x)-3\*x\*\*2, 2, 5, 20)

Bisection: findRoots(lambda x:math.exp(x)-3\*x\*\*2, 0, 5, 0.000001)

כפי שרואים מהטבלאות, הקבוע מתכנס ל- 0.910007 בערך (אבל בשיטת החציה לוקח יותר איטרציות).

נחלק ב- 1000 כדי לקבל את:

**µ =** 0.910007 / 1000 = **0.000910007**

1. נחשב את הקבוע k  
   בעזרת אינטרפולציה ממעלה שניה f(3.83) המתקבל מהטבלה

|  |  |
| --- | --- |
| y | x |
| -3.5 | 2 |
| 1.25 | 3 |
| 4.1 | 6 |

לאחר חישוב interpolation הנעזר בקוד חיצוני (ראה נספח א) נקבל:

**k = 3.74954**

1. לאחר חישוב כל הקבועים נוכל להתחיל לחשב את רמת הזיהום בכל ריבוע בשטח.  
   נשתמש בקוד שבנינו כאן: <https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/Main.py>

**סעיף ג' – הצגת הנתונים –**

TBA

**סעיף ד' – תוצאות –**

לאחר חישוב מטריצת המדידות D באמצעות הנוסחה שהובאה לנו במאמר

הגענו למדידות הבאות:

[[0.01495167 0.01209884 0.01033197 0.01209884]

[0.01209884 0.01495167 0.01209884 0.01033197] **D** =

[0.01033197 0.01209884 0.01495167 0.01209884]

[0.01209884 0.01033197 0.01209884 0.01495167]]

כעת נוכל לחשב את וקטור המשתנים שאינם ידועים לנו – C.

נשתמש ב-2 שיטות שנלמדו למציאת הווקטור, באמצעות כפל עם המטריצה ההופכית D^(-1)

ווקטור התוצאה M, ובאמצעות שיטת גאוס:

לפי M\* inverted matrix(D):

כדי לחשב כפל של מטריצה M עם המטריצה ההופכית של D, השתמשנו בספריית scipy

על מנת למצוא את ההופכית של D (ראה 'קוד שנעשה בו שימוש חוזר'), והשתמשנו

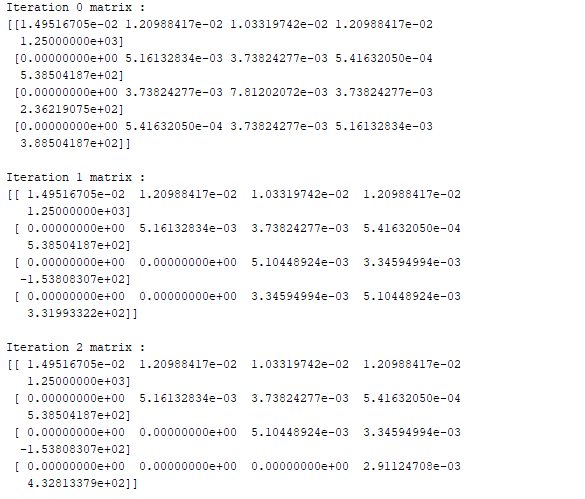
בכפל מטריצה בווקטור עם פונקציה *.dot* (גם כן ב- scipy).

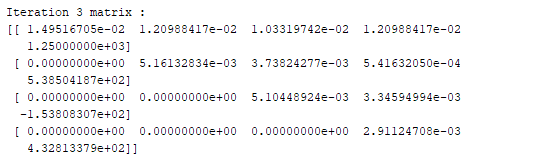
בסופו של דבר, התקבלה התוצאה הבאה:

***C*** *= (-95113.85256187, 181139.07459656, -127583.51949139, 148669.40766704)*

לפי :Gauss

בשיטת דירוג המטריצות של גאוס, הפונקציה מקבלת את המטריצה D עם וקטור התוצאה M ומחזירה וקטור סופי C, שלבי הדירוג מתוארים כך:

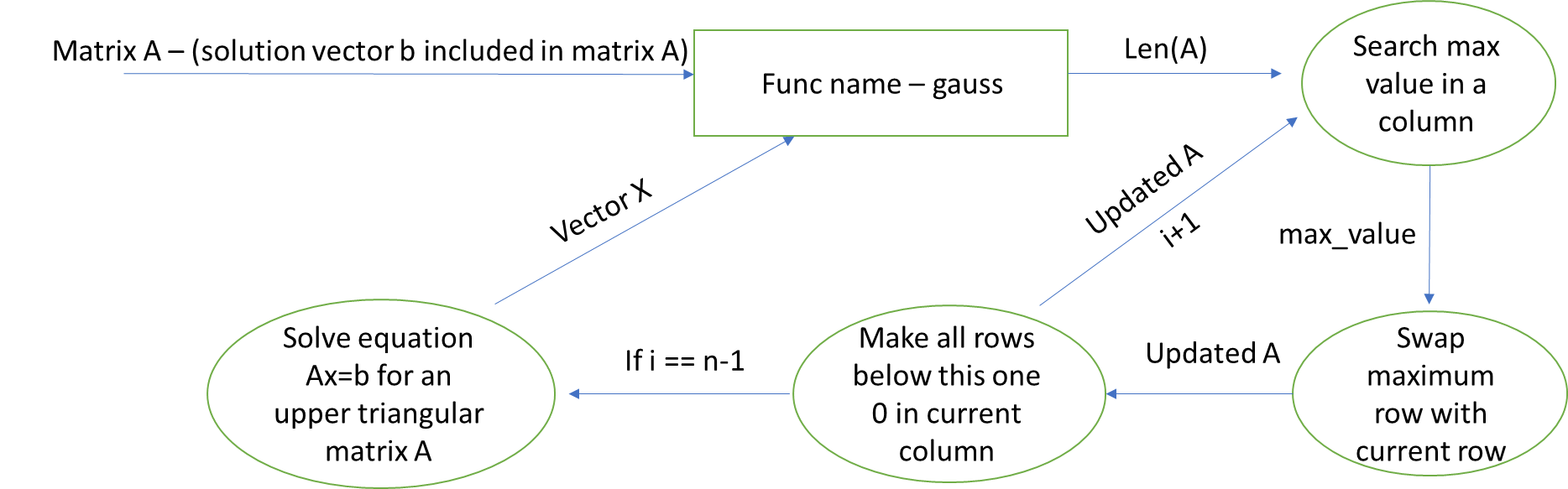




בסוף הדירוג, מתקבל וקטור C הבא:

***C*** *= (-95113.853, 181139.075, -127583.52, 148669.408)*

ניתן לראות כי רמת הדיוק שהתקבלה, שונה מהתוצאה שהתקבלה בשיטה הראשונה (inverted matrix), בגלל שהקנינו דיוק של 3 אחרי הנקודה, ע"י פונקציה מובנת round ()



(תרשים המתאר את שלבי הדירוג עפ"י הקוד של פונקציית גאוס)

**סעיף ה' – סיכום –**

TBA

**נספח א’ –**

קוד שלנו:

* [Gauss Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/GaussAlgo.py#L22)
* [Matrix Norma calculation](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/GaussAlgo.py#L4)
* [Matrix CondA calculation](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/GaussAlgo.py#L19)
* [Newton Repson Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/NewtonRepson.py)
* [Interpolation](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/GaussAlgo.py#L59)

קוד שנעשה בו שימוש חוזר:

SciPy (<https://www.scipy.org/>) and NumPy (<http://www.numpy.org/>)

* numpy.tril
* numpy.dot
* numpy.array
* numpy.linalg.inv
* scipy.linalg.lu
* [Jacobi Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/Jacobi.py) - <https://en.wikipedia.org/wiki/Jacobi_method#An_example_using_Python_and_Numpy>
* [Bisection Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/BisectionAlgorithm.py) - <http://code.activestate.com/recipes/578417-bisection-method-in-python/>
* [Secant Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/SecantAlgorithm.py) - <http://code.activestate.com/recipes/578420-secant-method-of-solving-equtions-in-python/>
* [Gauss-Seidel Algorithm](https://github.com/SCE-SWE-2018-G11/NumericAnalysis/blob/master/Gauss-Seidel.py) - <https://austingwalters.com/gauss-seidel-method/>