

## האקטון אנליזה נומרית:

### מגשים:

ליאור וקנין – 208574046  
פבל שוורצוב – 319270583  
אליאור קרצמן – 313565095  
ארתור אדינייב – 314307414  
מורדי דבח – 203507017

### הסבר על המאמר:

המאמר מציג בפנינו את אחד השימושים העיקריים בשיטת ניטור האוויר. משתמשים בעצם באיזוטופים רדיואקטיביים כדי להתחכות בעצם אחר הקרינה. הבעיה היא בחישוב המתמטי של הניטור כאשר מדובר במרחב  $3d$ . אז מנסים להתגבר על הבעיה באמצעות הערכה מספיק טובה של התפלגות הזיהום  $2d$  (בעצם על הקרקע).

### פתרון הבעיה :

נרצה להגיע למטריצה (רשת פריסה) ולמצוא את ערכיה כאשר כל נקודה  $(i,j)$  מרכזית תייצג את רמת הקרינה בתא בא היא נמצאת. ננסה לפענח את הסיטואציה ולמצוא ערכים אופטימליים לגודל  $N^2$  ולגודל של  $R$  כך שמהטריצה  $D$  לא תהיה מסובכת מדי לדירוג ושהפתרון שלה יתכנס ( ill condition). כדי שנוכל למצוא מטריצה הפיכה כדי שנוכל לפתור את הבעיה בעזרת פונקציות אלמנטריות.

## מסמך הסבר קוד להאקטון ומדריך למשתמש לכל אלגוריתם

שיטת החצייה – Bisection method:

שיטת החצייה מתאימה אך ורק לשורשים מריבוב אי זוגי. כלומר ש:

מספר קטן מאוד  $j$  כך ש-  $f(a-j)*f(a+j)<0$

שיטת החצייה שכתבנו, קודם כל בודקת אם השורש הוא מריבוב אי זוגי. אם כן, נמשיך באלגוריתם. אחרת, נעצור את התוכנית.

Input:

ax – point a

bx – point b

function f

epsilon – point of accuracy. If the answer is in this epsilon bounds, we will stop the function and return the value.

Output:

The function bisection method prints the a and b values along the algorithm. In the end of the algorithm the function prints the result value of the root.

האלגוריתם האיטרטיבי בודק כל פעם איפה השורש. בין נקודת האמצע לנקודה השמאלית או בין האמצע לנקודה הימנית. ולפי זה קובע גבולות חדשים.

אם התוצאה מספיק קרובה לרמת הדיוק שהגדרנו (אפסילון) נעצור ונחזיר את השורש אליו הגענו.

User guide:

First enter a function f in python format. (Example shown in picture)

Then enter a and b values:

```
Run bisection_method
C:\Python34\python.exe C:/Users/רן יאלי/PycharmProjects/Numerical/bisection_method.py
Enter a function f(x):
x**2-2
Enter range: a and b
a=1
b=2
aX: 1.0, bX: 2.0
aX: 1.0, bX: 1.5
aX: 1.25, bX: 1.5
aX: 1.375, bX: 1.5
aX: 1.375, bX: 1.4375
aX: 1.40625, bX: 1.4375
aX: 1.40625, bX: 1.421875
aX: 1.4140625, bX: 1.421875
aX: 1.4140625, bX: 1.41796875
aX: 1.4140625, bX: 1.416015625
aX: 1.4140625, bX: 1.4150390625
aX: 1.4140625, bX: 1.41455078125

The root is: 1.4141845703125

Process finished with exit code 0
|
```

## שיטת המיתר – Secant method

עבור שיטה זו דרושים שני ניחושים התחלתיים. שיטה זו היא שיטה איטרטיבית למציאת שורשים של פונקציה רציפה בין שני נקודות (שהם הניחושים ההתחלתיים). נעצור את האלגוריתם כאשר אנו מתקרבים לדיוק מספיק טוב (שאנו הגדרנו – אפסילון).

האלגוריתם:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

Input:

Function f

x0 – first point (guess)

x1 – second point (guess)

epsilon – point of accuracy. If the answer is in this epsilon bounds, we will stop the function and return the value.

nMax – number of maximum iterations.

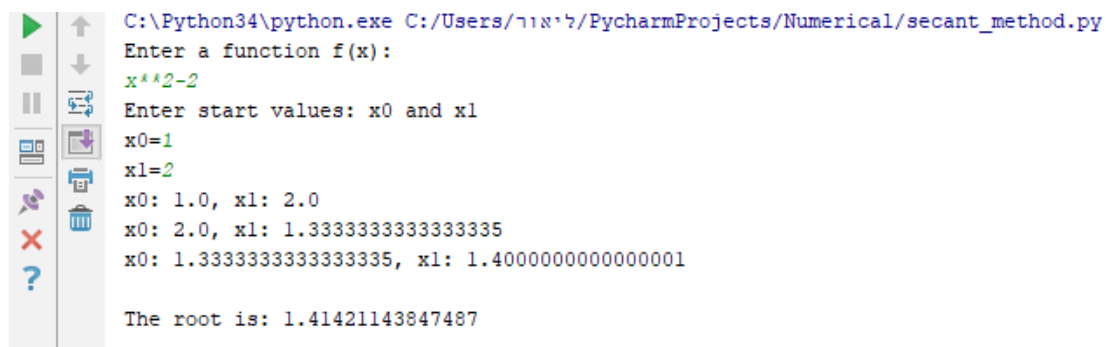
Output:

The function secant method prints the a and b values along the algorithm calculated according to the secant method algorithm. In the end of the algorithm the function prints the result value of the root.

User guide:

First enter a function f in python format. (Example shown in picture)

Then enter x0 and x1 values:



```
C:\Python34\python.exe C:/Users/ליאור/PycharmProjects/Numerical/secant_method.py
Enter a function f(x):
x**2-2
Enter start values: x0 and x1
x0=1
x1=2
x0: 1.0, x1: 2.0
x0: 2.0, x1: 1.3333333333333335
x0: 1.3333333333333335, x1: 1.4000000000000001

The root is: 1.41421143847487
```

## שיטת ניוטון-רפסון – Newton-Rapson method:

שיטת ניוטון-רפסון היא שיטה איטרטיבית למציאת שורשים של משוואה. שיטה זו עושה שימוש בנגזרת של נקודה. כלומר, הנגזרת צריכה להיות מוגדרת וצרי לחשב אותה.

חישבנו נגזרת לפונקציה נתונה והחזרנו פונקציה שהיא פונקציית הנגזרת שלה, לפי פתרון נומרי:

```
def derive(f):  
    """  
    This function return the numerical calculated derived function of f.  
    :param f: the function  
    :return: derived function  
    """  
    h = 10**-10  
    return lambda x: (f(x + h) - f(x))/float(h)
```

לפי הגזרה של נגזרת של פונקציה:

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

בפיתרון שלנו המשתנה ששואף לאפס שווה ל10 בחזקת מינוס 10 (מספר ממש קטן).

האלגוריתם:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Input:

For derive function:

function f

For newton-Raphson method:

function f

start value x0

epsilon – point of accuracy. If the answer in in this epsilon bounds, we will stop the function and return the value.

nMax – number of maximum iterations.

Output:

For derive function:

the derived function of the function f

For newton-Raphson method:

the algorithm returns false if we finished the while loop without getting close to epsilon.

otherwise, the algorithm returns the value of the root of the function that was

given.

Because, in this algorithm we divide from the value of  $f(x_n)$  the value of  $f'(x_n)$ , we check first that the  $f'(x_n)$  is different than zero.

User guide:

First enter a function  $f$  in python format. (Example shown in picture)

Then enter  $x_0$  start value:

```
C:\Python34\python.exe C:/Users/רורל/PycharmProjects/Numerical/newtonraphson_method.py
Enter a function f(x):
x**2-2
Enter x0 start value:
1
x0: 1.0, x1: 1.499999958629818
x0: 1.499999958629818, x1: 1.416666673561693

The root is: 1.416666673561693
```

### קירוב (אינטרפולציה) לינארית:

נשתמש במתודה זו כשנרצה לחשב ערך בין שני נקודות קרובות (יחסית) אחת לשנייה, ושהנקודות הן ידועות בגרף.

נפתור לפי האלגוריתם שלמדנו בהרצאה.

#### Input:

$x_1$  – value  $x_1$  of the known point  $x$  in the graph.

$x_2$  – value  $x_2$  of the known point  $x$  in the graph.

$y_1$  – value  $y_1$  of the known point  $y$  in the graph.

$y_2$  – value  $y_2$  of the known point  $y$  in the graph.

$x_f$  – the value of the  $x$  point in which we want to approximate its  $y$  value.

#### Output:

The linear approximation of the given values.

שיטת דירוג גאוס – Gauss elimination method:

הרעיון של האלגוריתם הוא לבצע רצף של פעולות אלמנטריות כדי להביא אותה לצורה מדורגת קנונית.

URL: [https://github.com/WaizungTaam/Gaussian-Elimination/blob/master/gaussian\\_eliminate.py](https://github.com/WaizungTaam/Gaussian-Elimination/blob/master/gaussian_eliminate.py)

השתמשנו בספריית  
numpy  
כדי לייצג מטריצות.

Input:

Matrix A, size of  $N \times N$

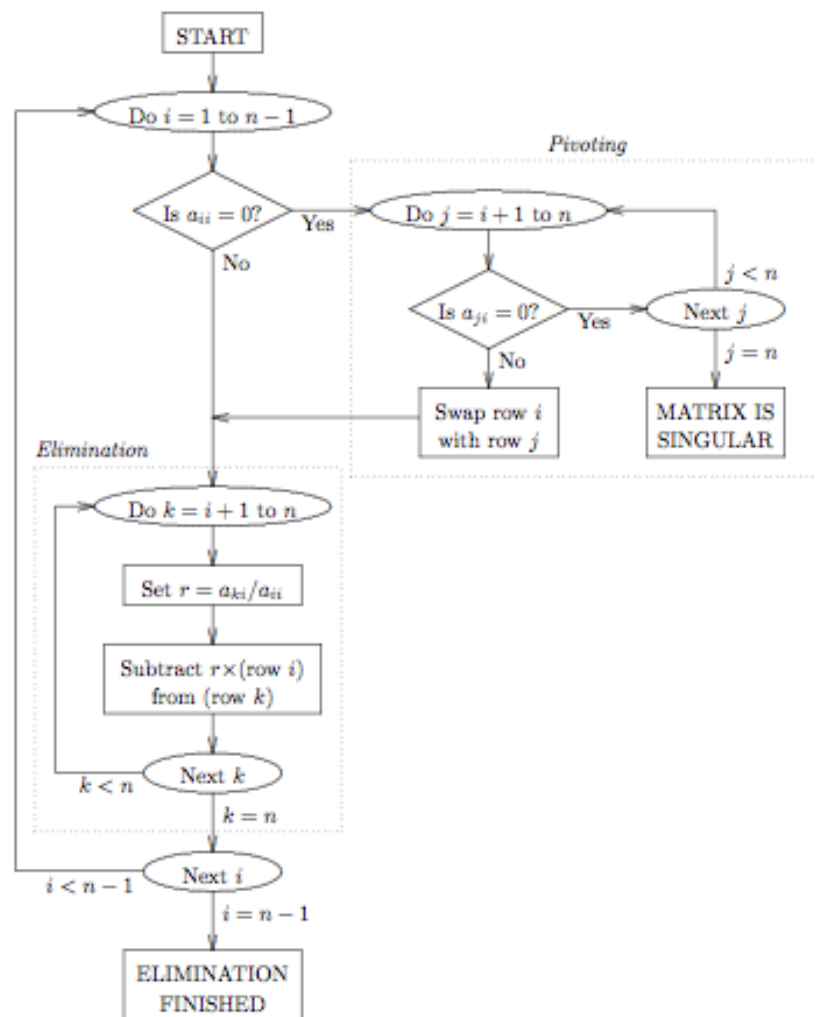
Output:

The algorithm prints the canonical eliminated matrix A with the result vector as vector b.

The algorithm returns and prints the result vector b.



gauss flowchart:



### קירוב (אינטרפולציה) פולינומיאלית:

קירוב פולינומיאלי מקבל מילון (בפייטון) בטור טבלה.  
מהמילון אנו יוצרים מטריצה בגודל של המילון. מערכי ה  
value  
אנו יוצרים וקטור פיתרונות.

את המטריצה מדרגים לקבלת מטריצה מדורגת קנונית ואת ווקטור התוצאות אנו לוקחים ויוצרים  
ממנו פולינום. באמצעות ספרייה מוכנה בפייטון.

Input:

Dictionary in python with keys and values.

Output:

Polynomial of the approximation.

User guide:

Enter a dictionary as:

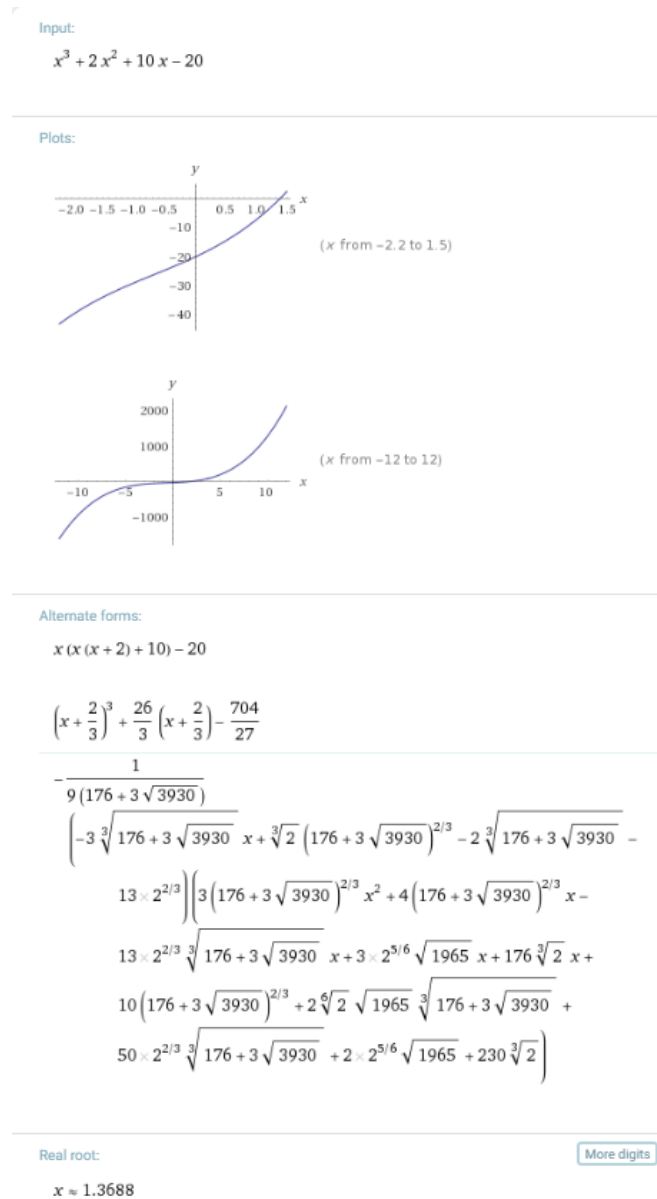
{key1:val1, key2:val2, ..., keyn:valn}

## פיתרון להאקטון:

### גודל C:

הגודל C מתקבל מתוך חישוב של מחצית השורש האמיתי של משוואת לינארדו:  
נפעיל שתי שיטות למציאת שורשים.

לפי wolfram alpha השורש הוא  $x=1.3688$



## שיטת החצייה – bisection method:

```
"C:\Users\Lior Vaknin\PycharmProjects\Numerial\venv\Scripts\python.exe" "C:/Users/Lior Vaknin/PycharmProjects/Numerial/bisection_method.py"
Enter a function f(x):
x**3+2*x**2+10*x-20
Enter range: a and b
a=1
b=2
aX: 1.0, bX: 2.0
aX: 1.0, bX: 1.5
aX: 1.25, bX: 1.5
aX: 1.25, bX: 1.375
aX: 1.3125, bX: 1.375
aX: 1.34375, bX: 1.375
aX: 1.359375, bX: 1.375
aX: 1.3671875, bX: 1.375
aX: 1.3671875, bX: 1.37109375
aX: 1.3671875, bX: 1.369140625
aX: 1.3681640625, bX: 1.369140625
aX: 1.36865234375, bX: 1.369140625

The root is: 1.3687744140625
```

השורש הוא:  $x=1.36877$   
בקירוב של  $\epsilon=10^{-4}=0.0001$

## שיטת ניוטון-רפסון – Newton-Raphson:

```
"C:\Users\Lior Vaknin\PycharmProjects\Numerial\venv\Scripts\python.exe"
Enter a function f(x):
x**3+2*x**2+10*x-20
Enter x0 start value:
1
x0: 1.0, x1: 1.4117646718127912
x0: 1.4117646718127912, x1: 1.3693364593869966

The root is: 1.3693364593869966
```

השורש הוא:  $x=1.3693$   
הקירוב של הנגזרת  $h=10^{-10}$  (המספר שקרוב מאוד לאפס – שואל לאינסוף)  
בקירוב של  $\epsilon=10^{-4}=0.0001$

## שיטת המיתר – secant method:

```
"C:\Users\Lior Vaknin\PycharmProjects\Numerial\venv\Scripts\python.exe"
Enter a function f(x):
x**3+2*x**2+10*x-20
Enter start values: x0 and x1
x0=1
x1=2
x0: 1.0, x1: 2.0
x0: 2.0, x1: 1.3043478260869565
x0: 1.3043478260869565, x1: 1.3579123046578667

The root is: 1.3688074597219246
```

השורש הוא:  $x=1.3688$   
בקירוב של  $\epsilon=10^{-4}=0.0001$

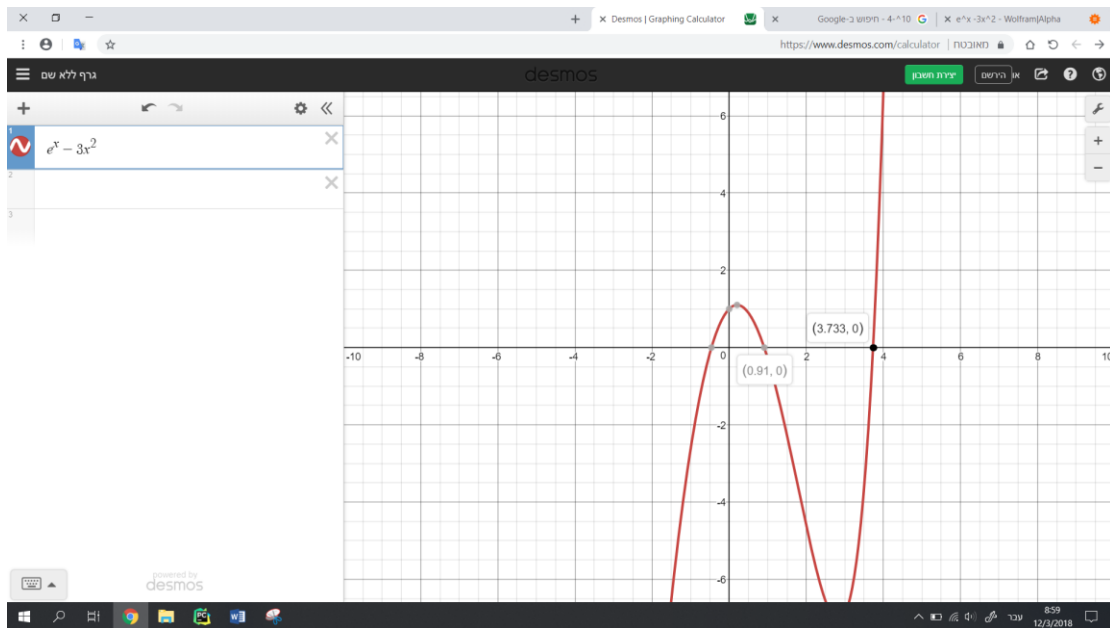
ולכן הגודל  $c$  שווה:

$$c = x/2 = 0.6844$$

גודל של מקדם הבליעה:

מקדם הבליעה שווה לאלפית השורש החיובי של המשוואה  $e^x - 3x^2$

לפי Desmos:



השורש האמיתי הוא:  $x=3.733$

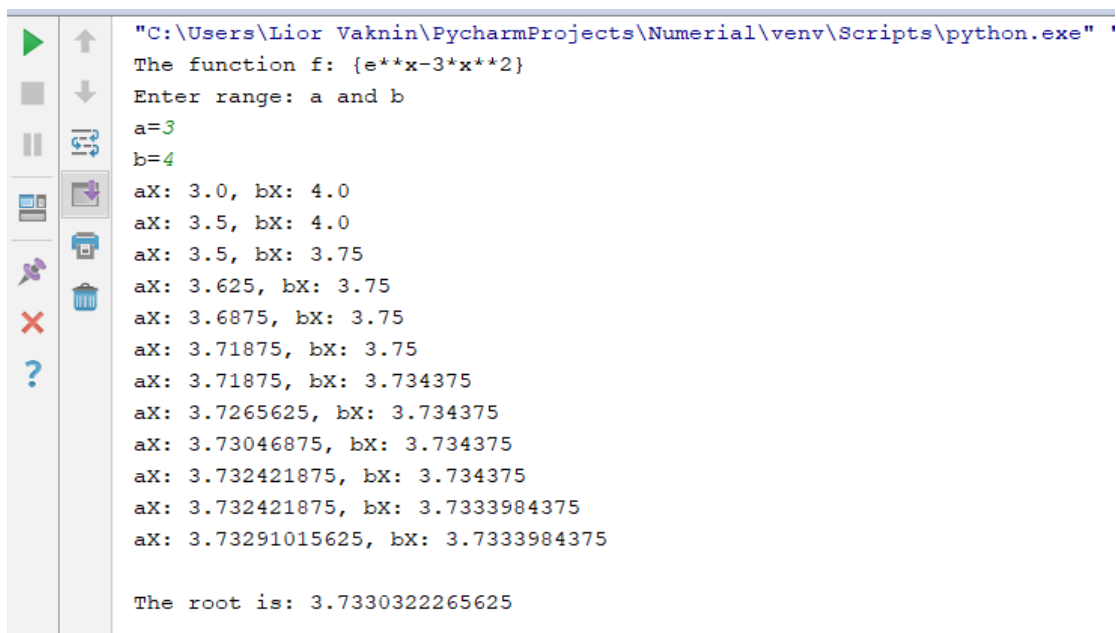
לפי שיטת המיתר – secant method:

```
"C:\Users\Lior Vaknin\PycharmProjects\Numerical\venv\Scripts\python.exe"
The function f: {e**x-3*x**2}
Enter start values: x0 and x1
x0=3
x1=4
x0: 3.0, x1: 4.0
x0: 4.0, x1: 3.5117043624757907
x0: 3.5117043624757907, x1: 3.680658256169178
x0: 3.680658256169178, x1: 3.745598502519386
x0: 3.745598502519386, x1: 3.732460650522579

The root is: 3.7330790326819776
```

השורש הוא  $x=3.733$   
לפי קירוב של  $\epsilon=10^{-4}=0.0001$

לפי שיטת החצייה – bisection method:



```
"C:\Users\Lior Vaknin\PycharmProjects\Numerical\venv\Scripts\python.exe"
The function f: {e**x-3*x**2}
Enter range: a and b
a=3
b=4
aX: 3.0, bX: 4.0
aX: 3.5, bX: 4.0
aX: 3.5, bX: 3.75
aX: 3.625, bX: 3.75
aX: 3.6875, bX: 3.75
aX: 3.71875, bX: 3.75
aX: 3.71875, bX: 3.734375
aX: 3.7265625, bX: 3.734375
aX: 3.73046875, bX: 3.734375
aX: 3.732421875, bX: 3.734375
aX: 3.732421875, bX: 3.733984375
aX: 3.73291015625, bX: 3.733984375

The root is: 3.7330322265625
```

השורש הוא  $x=3.7330$   
לפי קירוב של  $\epsilon=10^{-4}=0.0001$

ולכן מקדם הבליעה הוא אלפית השורש שיצא לנו:

$$x/1000 = 3.7330/1000 = 0.003733 = \text{מקדם הבליעה}$$

3) קירוב פולונומיאלי: הכנסנו למילון את ערכי הטבלה :

{2:-3.5,3:1.25,6:0.05}

ופונקציה שמחשבת פתרון של מטריצה, לפנו' **polynomial\_approximation**  
והפונקציה חישה את מקדמי הפולינום שאליו קירבנו והחזירה לנו פולינום עם מקדמים אלו.

פתרון עם פונקציה פתירת מטריצה **gaussian\_eliminate\_result**:

```
polynomial_approx  polynomial_halving_method
35 if __name__ == '__main__':
36     # solver = addKwargs(Matrix.iterSOR,n=100,w=1.5)
37     p1 = polynomial_approximation({2:-3.5, 3:1.25, 6:0.05},matrixSolver=gaussian_eliminate_result_adapter)
38     p1 = Polynom(p1)
39     print("f(3.83) = ", p1.eval(3.83))

<terminated> polynomial_approx.py [D:\Programs\Programming\Python 3.6.5\python.exe]
0.0 1.5 0.75 | 1.2375
0.0 0.0 0.222222222222221 | -4.605555555555556

36.0 6.0 0.0 | 20.775000000000016
0.0 1.5 0.0 | 16.781250000000001
0.0 0.0 0.222222222222221 | -4.605555555555556

36.0 0.0 0.0 | -46.350000000000002
0.0 1.5 0.0 | 16.781250000000001
0.0 0.0 0.222222222222221 | -4.605555555555556

1.0 0.0 0.0 | -1.2875000000000005
0.0 1.5 0.0 | 16.781250000000001
0.0 0.0 0.222222222222221 | -4.605555555555556

1.0 0.0 0.0 | -1.2875000000000005
0.0 1.0 0.0 | 11.187500000000007
0.0 0.0 0.222222222222221 | -4.605555555555556

Final matrix A:

1.0 0.0 0.0 | -1.2875000000000005
0.0 1.0 0.0 | 11.187500000000007
0.0 0.0 1.0 | -20.725000000000012

Vector x (result): [-1.2875000000000005, 11.187500000000007, -20.725000000000012]
guess error: 1.136590821460004e-14
f(3.83) = 3.2369162500000073
```

פתרון עם פונ' פתירת מטריצה Matrix.iterGaussSeidel:

```
polynomial_approx.py
33 if __name__ == '__main__':
34     solver = addKwargs(Matrix.iterGaussSeidel,n=25)
35     print(polynomial_approximation({2:-3.5, 3:1.25, 6:0.05},matrixSolver=solver))
36

Console
<terminated> polynomial_approx.py [D:\Programs\Programming\Python 3.6.5\python.exe]
guess #3: [[-35.229210000000004], [79.729210000000007], [297.929]]
guess #4: [[-278.22083333333333], [515.77083333333334], [6921.375]]
guess #5: [[-1989.1041666666667], [3660.6041666666665], [49644.175]]
guess #6: [[-14242.220833333333], [26179.020833333336], [355645.875]]
guess #7: [[-102001.85416666667], [187457.35416666666], [2547322.675]]
guess #8: [[-730560.2208333333], [1342573.5208333335], [18244726.875]]
guess #9: [[-5232469.354166667], [9615832.854166666], [130673899.675]]
guess #10: [[-37476392.22083333], [68871210.52083333], [935922856.8749999]]
guess #11: [[-268416320.35416663], [493274675.8541666], [6703339477.6749999]]
guess #12: [[-1922472208.220833], [3532970132.520833], [48011178700.875]]
guess #13: [[-13769279742.354166], [25304112993.854168], [343869392761.675]]
guess #14: [[-98619404688.22083], [181235083144.5208], [2462888069908.8745]]
guess #15: [[-706339559050.354], [1298055987181.854], [17639888202721.67]]
guess #16: [[-5059000044272.22], [9297037398576.52], [126341777202340.86]]
guess #17: [[-36233962999874.35], [66587963265509.84], [904894888402417.5]]
guess #18: [[-259517703733360.2], [476921481732608.44], [6481108444005316.0]]
guess #19: [[-1858737851867634.0], [3415844074267797.5], [4.641949822162805e+16]]
guess #20: [[-1.331279659254091e+16], [2.446522370374672e+16], [3.324693351089925e+17]]
guess #21: [[-9.534994562912149e+16], [1.7522672518436698e+17], [2.3812376915421716e+18]]
guess #22: [[-6.829227854777265e+17], [1.2550224592524554e+18], [1.705508552168342e+19]]
guess #23: [[-4.891282610047082e+18], [8.988819322913442e+18], [1.2215325802421433e+20]]
guess #24: [[-3.5032724167510303e+19], [6.4380419827792806e+19], [8.748955510636142e+20]]
guess #25: [[-2.5091409767979994e+20], [4.611104426848618e+20], [6.266244860363628e+21]]
done in iteration number : 25
guess error: 6.184809355014152e+21 , result guess:
[[-2.50914098e+20]
 [ 4.61110443e+20]
 [ 6.26624486e+21]]
2
-2.509e+20 x + 4.611e+20 x + 6.266e+21[
```



## פיתרון מטריצת D:

```
M:
[[1250]
 [1550]
 [1100]
 [1400]]
D:
[[0.00845397 0.00524882 0.00524882 0.00362924]
 [0.00524882 0.00845397 0.00362924 0.00524882]
 [0.00524882 0.00362924 0.00845397 0.00524882]
 [0.00362924 0.00524882 0.00524882 0.00845397]]
```

חישוב רמת זיהום:

```
def getDMatrix():
    c = 0.6844
    k=3.23691625
    u = 0.003733
    math.e
    def getRadius(i,j):
        radius = [[150,180,180,206.155],[180,150,206.155,180],[180,206.155,150,180],[206.155,180,180,150]]
        return radius[i][j]

    def getD(i,j):
        return (c*(1+k*150)*math.e**(-u*getRadius(i, j)))/getRadius(i, j)**2

    result = [[getD(i,j) for j in range(4)] for i in range(4)]
    return np.matrix(result)

def CalcC(solver):
    M = Matrix([[x] for x in [1250,1550,1100,1400]])
    D = getDMatrix()
    print("M:\n",M)
    print("D:\n",D)
    return solver(D,M)

if __name__ == '__main__':
    solver = addKwargs(Matrix.iterGaussSeidel,n=50)
    # solver = gaussian_eliminate_result_adapter
    print("vector c: ", CalcC(solver).tolist())
```

calcC מייצר את ווקטור M ומטריצת D לפי המשוואה 1.3 מהמסמך ופותרת את C בהתאם לsolver שהוא פונ' מציאת פתרון למטריצה.

פתרון לפי שיטת gauss elim:

```
<terminated> main.py [D:\Programs\Programming\Python 3.6.5\python.exe]
0.008453967055416407 0.0 0.0 4.336808689942018e-19 | 364.6461831018804
0.0 0.0029955298629080747 0.0 0.0 | 315.467705522975
0.0 0.0 0.00482472744608122 0.0 | 58.10566639163997
0.0 0.0 0.0 -0.006821264759487217 | -506.29462277999477

1.0 0.0 0.0 5.1299096170045397e-17 | 43133.144559423585
0.0 0.0029955298629080747 0.0 0.0 | 315.467705522975
0.0 0.0 0.00482472744608122 0.0 | 58.10566639163997
0.0 0.0 0.0 -0.006821264759487217 | -506.29462277999477

1.0 0.0 0.0 5.1299096170045397e-17 | 43133.144559423585
0.0 1.0 0.0 0.0 | 105312.82275941574
0.0 0.0 0.00482472744608122 0.0 | 58.10566639163997
0.0 0.0 0.0 -0.006821264759487217 | -506.29462277999477

1.0 0.0 0.0 5.1299096170045397e-17 | 43133.144559423585
0.0 1.0 0.0 0.0 | 105312.82275941574
0.0 0.0 1.0 0.0 | 12043.305459427565
0.0 0.0 0.0 -0.006821264759487217 | -506.29462277999477

Final matrix A:

1.0 0.0 0.0 5.1299096170045397e-17 | 43133.144559423585
0.0 1.0 0.0 0.0 | 105312.82275941574
0.0 0.0 1.0 0.0 | 12043.305459427565
0.0 0.0 0.0 1.0 | 74222.98365941965

Vector x (result): [43133.144559423585, 105312.82275941574, 12043.305459427565, 74222.98365941965]
guess error: 0.0
vector c: [43133.144559423585, 105312.82275941574, 12043.305459427565, 74222.98365941965]
```

פתרון לפי שיטת gauss seidel:

```
Console
<terminated> main.py [D:\Programs\Programming\Python 3.6.5\python.exe]
guess #10: [[42442.46953591368], [105598.2226095677], [12583.358543555729], [74006.9872974825]]
guess #11: [[42713.371368337044], [105475.71148340765], [12368.108681792066], [74100.39645121596]]
guess #12: [[42882.977111866276], [105404.81888356444], [12235.244336681506], [74154.0923082145]]
guess #13: [[42986.432418902084], [105364.28629886082], [12155.074256365242], [74184.62023477105]]
guess #14: [[43048.267651969945], [105341.3573144627], [12107.571997523759], [74201.80338903314]]
guess #15: [[43084.619687773535], [105328.51135370771], [12079.848286606753], [74211.38617165067]]
guess #16: [[43105.69435737601], [105321.37868309385], [12063.876000392811], [74216.68411769539]]
guess #17: [[43117.765160431634], [105317.45175397804], [12054.778068503678], [74219.5889401703]]
guess #18: [[43124.604885841894], [105315.3073438614], [12049.648551338989], [74221.16885048994]]
guess #19: [[43128.44280463728], [105314.14564712421], [12046.783492030856], [74222.021347084]]
guess #20: [[43130.57692587367], [105313.52129842843], [12045.197219110141], [74222.47769032134]]
guess #21: [[43131.75352959108], [105313.18842807566], [12044.326270217882], [74222.71999667565]]
guess #22: [[43132.39692444913], [105313.01241588741], [12043.851925685105], [74222.84757810735]]
guess #23: [[43132.745941628775], [105312.92014336542], [12043.595631838449], [74222.91416152989]]
guess #24: [[43132.933772244694], [105312.87221073067], [12043.458250929492], [74222.94858259398]]
guess #25: [[43133.03405115023], [105312.84755636021], [12043.385203702066], [74222.9661934526]]
guess #26: [[43133.08715092763], [105312.8350129842], [12043.346686369377], [74222.97510009876]]
guess #27: [[43133.11502943181], [105312.82870944795], [12043.326553634917], [74222.9795455114]]
guess #28: [[43133.12953453633], [105312.82558650263], [12043.316128484605], [74222.98173016304]]
guess #29: [[43133.137008287784], [105312.82406534452], [12043.310784893667], [74222.98278384299]]
guess #30: [[43133.14081806765], [105312.82333973984], [12043.308076813957], [74222.98328019891]]
guess #31: [[43133.14273685748], [105312.82300281074], [12043.306721963483], [74222.9835068491]]
done in iteration number : 31
guess error: 3.9368496800307184e-06 , result guess:
[[ 43133.14368993]
 [105312.82285198]
 [ 12043.30605426]
 [ 74222.9836059 ]]
vector c: [[43133.14368993351], [105312.82285198335], [12043.306054255878], [74222.98360590306]]
```

**וקטור התוצאה הסופית:**

vector c: [[43133.14368993351], [105312.82285198335], [12043.306054255878], [74222.98360590306]]