Numerical Analysis Project 2 Report

Hagar Omar Abouroumia [5943]

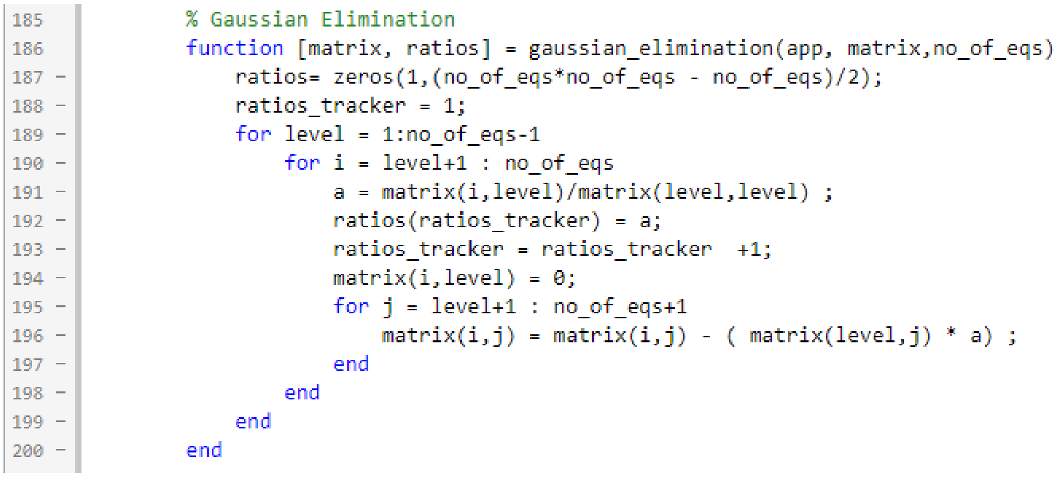
Karim M. Elsayad [6023]

Contents:

- Algorithms for each method
- GUI
- Sample runs

Algorithms for each method:

Gaussian Elimination



```
185         % Gaussian Elimination
186         function [matrix, ratios] = gaussian_elimination(app, matrix,no_of_eqs)
187 -           ratios= zeros(1,(no_of_eqs*no_of_eqs - no_of_eqs)/2);
188 -           ratios_tracker = 1;
189 -           for level = 1:no_of_eqs-1
190 -               for i = level+1 : no_of_eqs
191 -                   a = matrix(i,level)/matrix(level,level) ;
192 -                   ratios(ratios_tracker) = a;
193 -                   ratios_tracker = ratios_tracker  +1;
194 -                   matrix(i,level) = 0;
195 -                   for j = level+1 : no_of_eqs+1
196 -                       matrix(i,j) = matrix(i,j) - ( matrix(level,j) * a) ;
197 -                   end
198 -               end
199 -           end
200 -       end
```

- the function takes the matrix and the number of equations as inputs
- it returns the output matrix and ratios that will be used later in LU decomposition algorithm.

Line 187: we initialize the ratio array; the ratio array will contain the number we calculate for each row to get the new value.

Line 188: used to track the index of the ratio array

Line 189: the outer loop, which iterates on each column. Starting from the first column to before the last column.

Line 190: inner for loop, which iterates on each row, starting from the second row till the last row. We did not include the first row because it is the pivot equation

Line 191: calculate the number ‘a’ that will be used to get the new values of the elements in the current row.

Line 192: save the number ‘a’ in the ratios array

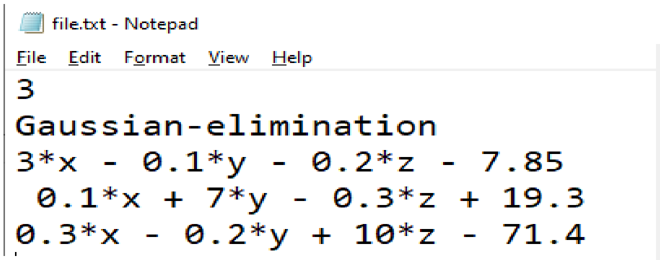
Line 193: increment the current index of the ratios array.

Line 194: set the element indicates by matrix(i , level) to zero.

Line 195: for loop to iterate over the elements in the current row to calculate their new values.

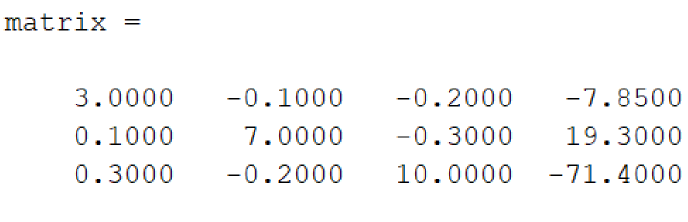
Example for elaboration:

Input:



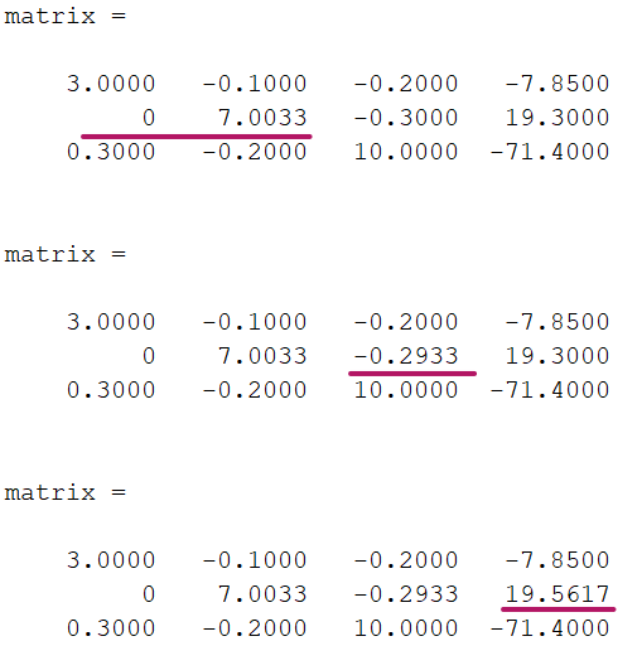
```
3
Gaussian-elimination
3*x - 0.1*y - 0.2*z - 7.85
 0.1*x + 7*y - 0.3*z + 19.3
0.3*x - 0.2*y + 10*z - 71.4
```

Corresponding Matrix:



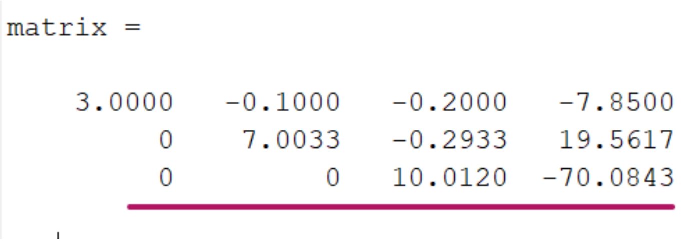
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
    0.1000    7.0000   -0.3000   19.3000
    0.3000   -0.2000   10.0000  -71.4000
```

First Iteration



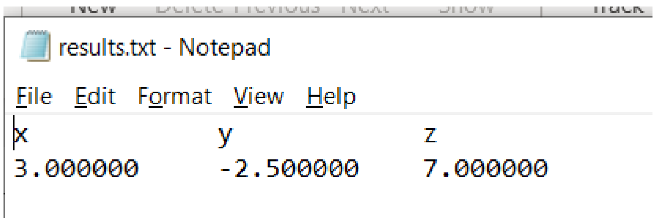
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.3000   19.3000
    0.3000   -0.2000   10.0000  -71.4000


matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.2933   19.3000
    0.3000   -0.2000   10.0000  -71.4000


matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.2933   19.5617
    0.3000   -0.2000   10.0000  -71.4000
```

Second iteration and the final Matrix



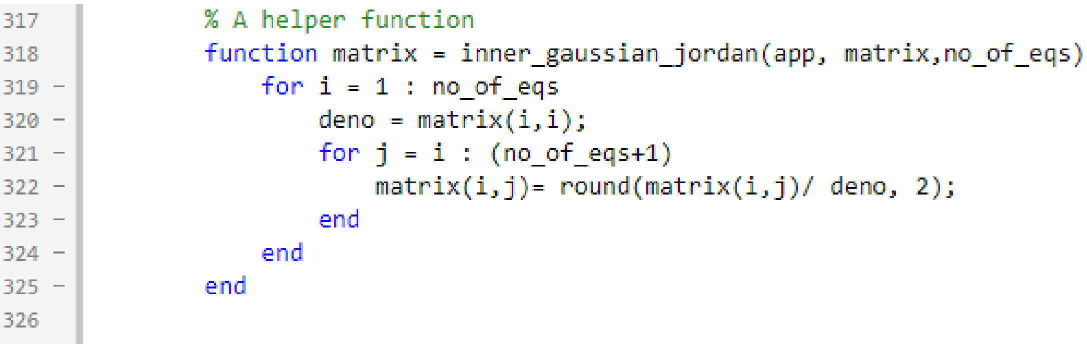
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.2933   19.5617
         0         0   10.0120  -70.0843
```

Output:



```
x              y              z
3.000000       -2.500000      7.000000
```

Gaussian Jordan



```
317           % A helper function
318           function matrix = inner_gaussian_jordan(app, matrix,no_of_eqs)
319 -             for i = 1 : no_of_eqs
320 -                 deno = matrix(i,i);
321 -                 for j = i : (no_of_eqs+1)
322 -                     matrix(i,j)= round(matrix(i,j)/ deno, 2);
323 -                 end
324 -             end
325 -         end
326
```

- inner gaussian Jordan is used as a part of the gaussian Jordan to get the new values for each element in the matrix.

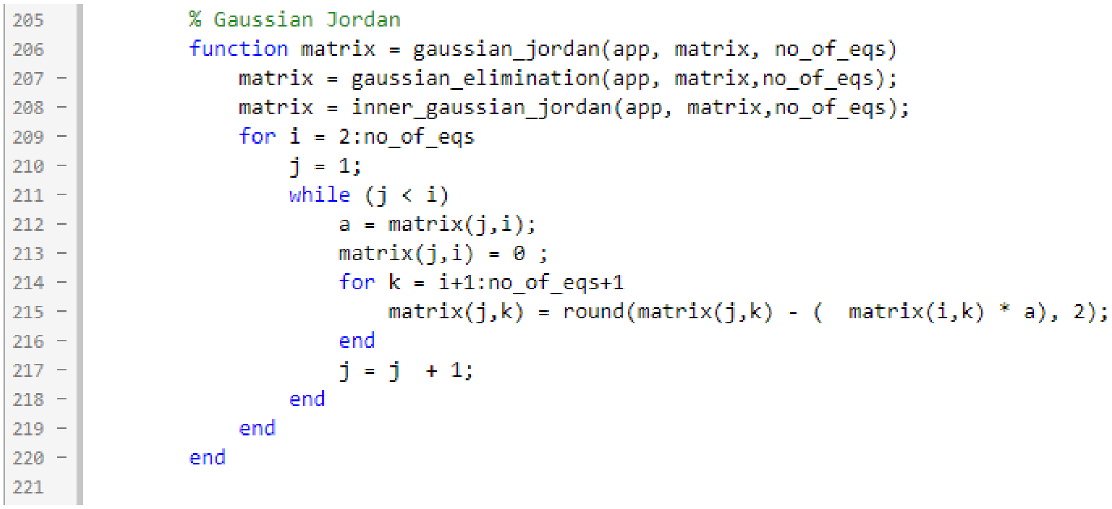
Line 319: we loop from the first row till the last row

Line 320: we get the element indicated by matrix(i,i) as the denominator

Line 321: inner for loop to iterate through the elements in the current row

Line 322: we divide each element in the current row by the denominator

- in that way we will set all the diagonal elements to 1



```
205           % Gaussian Jordan
206           function matrix = gaussian_jordan(app, matrix, no_of_eqs)
207 -             matrix = gaussian_elimination(app, matrix,no_of_eqs);
208 -             matrix = inner_gaussian_jordan(app, matrix,no_of_eqs);
209 -             for i = 2:no_of_eqs
210 -                 j = 1;
211 -                 while (j < i)
212 -                     a = matrix(j,i);
213 -                     matrix(j,i) = 0 ;
214 -                     for k = i+1:no_of_eqs+1
215 -                         matrix(j,k) = round(matrix(j,k) - (  matrix(i,k) * a), 2);
216 -                     end
217 -                     j = j  + 1;
218 -                 end
219 -             end
220 -         end
221
```

- gaussian Jordan algorithm uses the gaussian elimination algorithm and a helper function called inner gaussian Jordan.

Line 207: we use the gaussian elimination as it is the first step in the algorithm

Line 208: we will send the current matrix to inner gaussian Jordan as was discussed above.

Line 209: outer for loop to iterate on each column starting from the second column

Line 210: we will initialize i = 1, as we need to iterate in each row starting from first till end

Line 212: we will get ‘a’ which will be used to get the new values for the current row

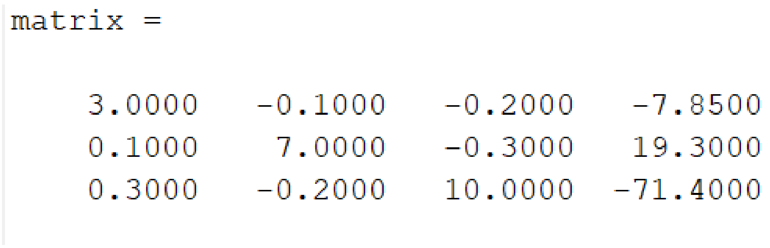
Line 213: we will set the element indicates by matrix(j,i) to 0.

Line 214: inner for loop on each element in the current row

Line 215: get the new values of each element.

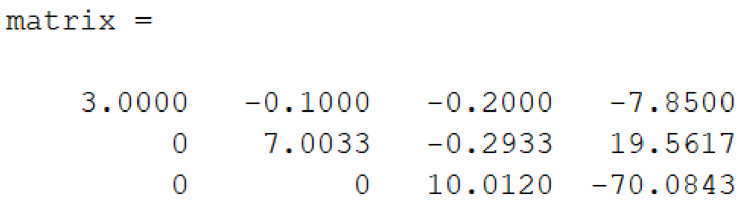
Example for elaboration:

Original Matrix



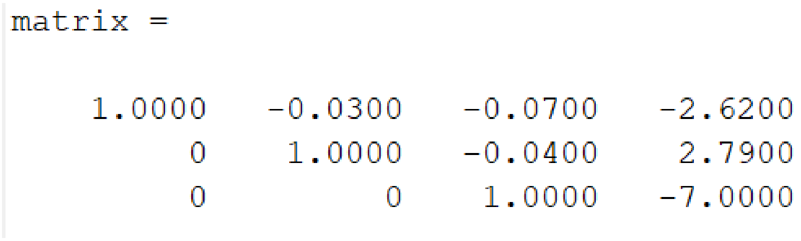
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
    0.1000    7.0000   -0.3000   19.3000
    0.3000   -0.2000   10.0000  -71.4000
```

Matrix after Gaussian elimination



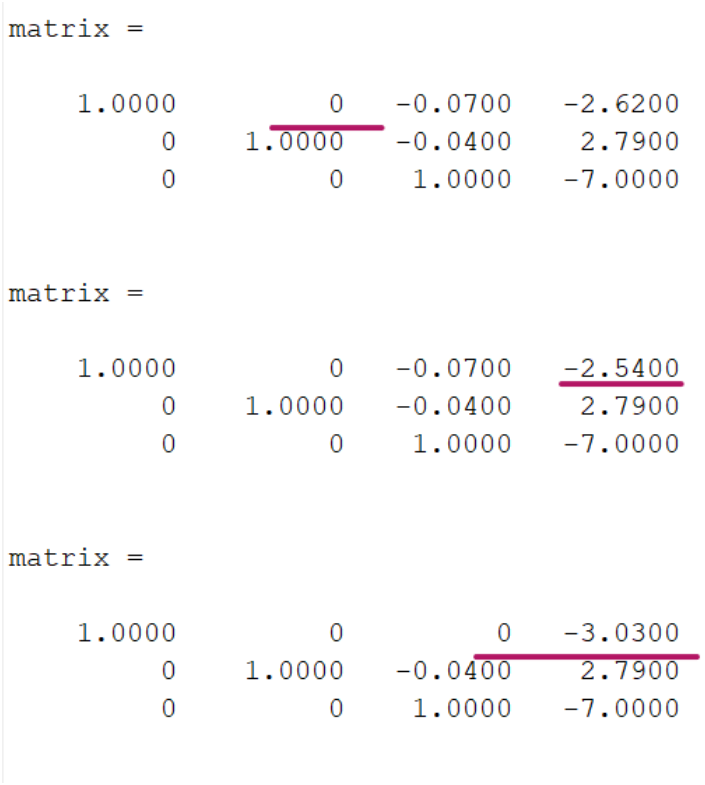
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.2933   19.5617
         0         0   10.0120  -70.0843
```

Matrix after Inner Gaussian Jordan



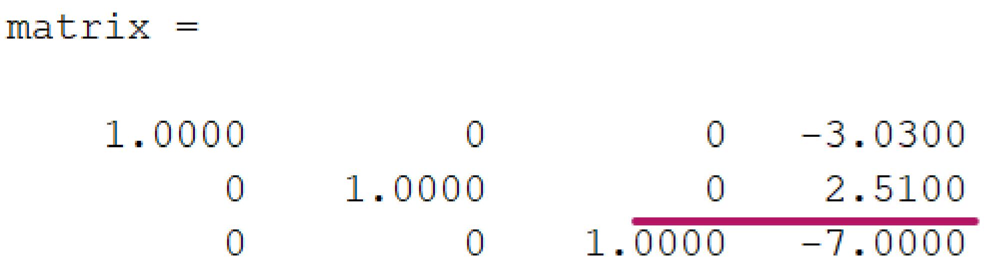
```
matrix =

    1.0000   -0.0300   -0.0700   -2.6200
         0    1.0000   -0.0400    2.7900
         0         0    1.0000   -7.0000
```

First iteration



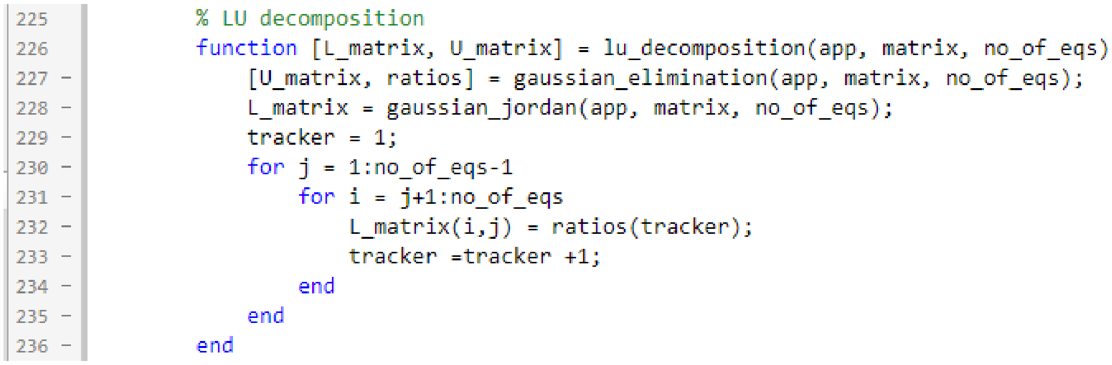
```
matrix =

    1.0000         0   -0.0700   -2.6200
         0    1.0000   -0.0400    2.7900
         0         0    1.0000   -7.0000


matrix =

    1.0000         0   -0.0700   -2.5400
         0    1.0000   -0.0400    2.7900
         0         0    1.0000   -7.0000


matrix =

    1.0000         0         0   -3.0300
         0    1.0000   -0.0400    2.7900
         0         0    1.0000   -7.0000
```

Second Iteration and Final Iteration



```
matrix =

    1.0000         0         0   -3.0300
         0    1.0000         0    2.5100
         0         0    1.0000   -7.0000
```

LU Decomposition



```
225          % LU decomposition
226          function [L_matrix, U_matrix] = lu_decomposition(app, matrix, no_of_eqs)
227 -            [U_matrix, ratios] = gaussian_elimination(app, matrix, no_of_eqs);
228 -            L_matrix = gaussian_jordan(app, matrix, no_of_eqs);
229 -            tracker = 1;
230 -            for j = 1:no_of_eqs-1
231 -                for i = j+1:no_of_eqs
232 -                    L_matrix(i,j) = ratios(tracker);
233 -                    tracker =tracker +1;
234 -                end
235 -            end
236 -        end
```

Line 227: we will call the gaussian elimination function just to get the ratios

Line 228: we will call the gaussian Jordan as it was discussed above

Line 229: initialize the index of the ratios array to 1

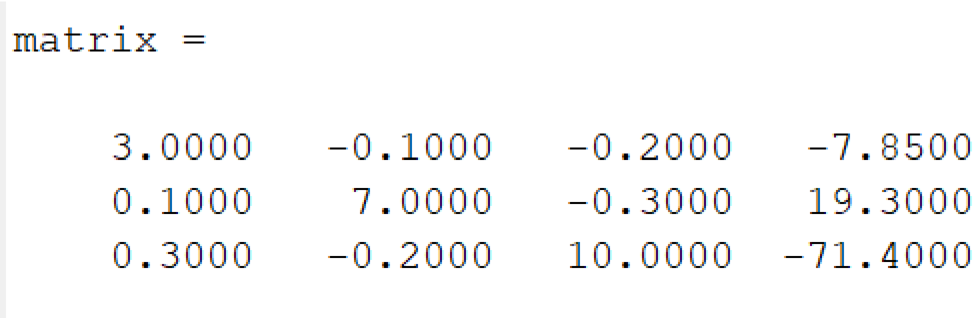
Line 230: outer for loop to iterate on the elements in each column

Line 231: inner for loop to iterate on the elements in each row

- in that way we got the L matrix

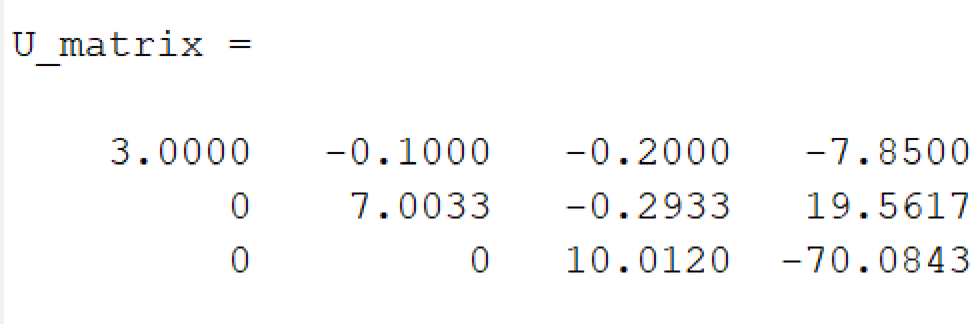
Example for elaboration:

Original Matrix



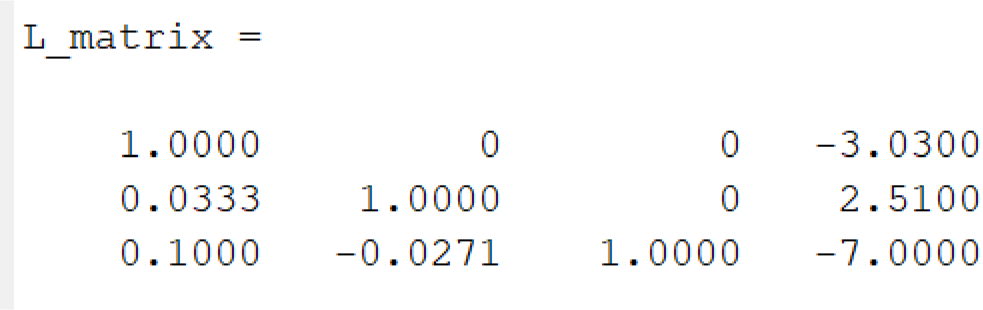
```
matrix =

    3.0000   -0.1000   -0.2000   -7.8500
    0.1000    7.0000   -0.3000   19.3000
    0.3000   -0.2000   10.0000  -71.4000
```

U matrix from gaussian elimination



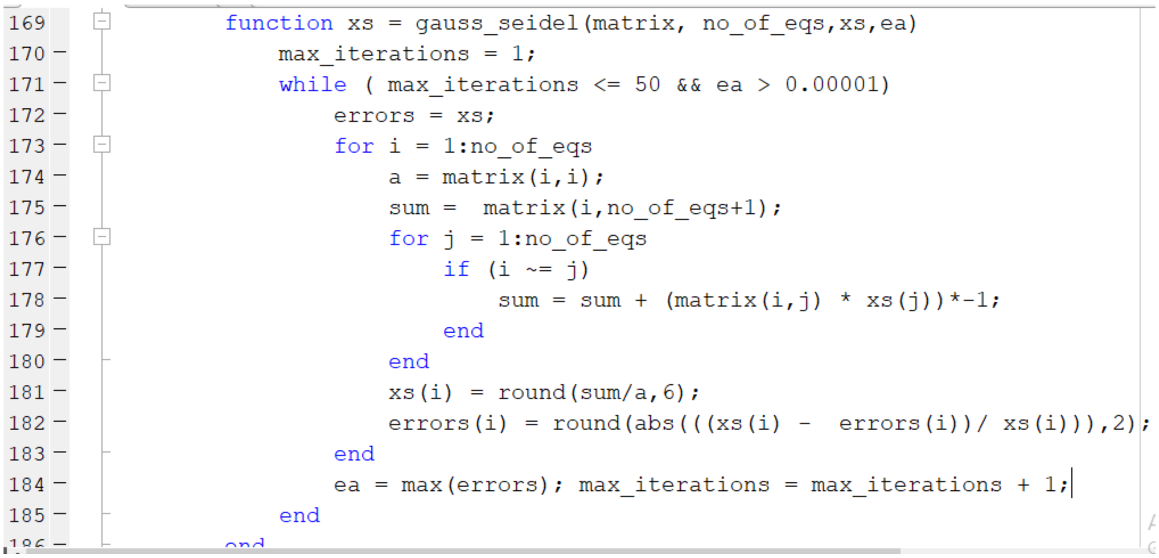
```
U_matrix =

    3.0000   -0.1000   -0.2000   -7.8500
         0    7.0033   -0.2933   19.5617
         0         0   10.0120  -70.0843
```

L matrix



```
L_matrix =

    1.0000         0         0   -3.0300
    0.0333    1.0000         0    2.5100
    0.1000   -0.0271    1.0000   -7.0000
```

Gaussian Seidel



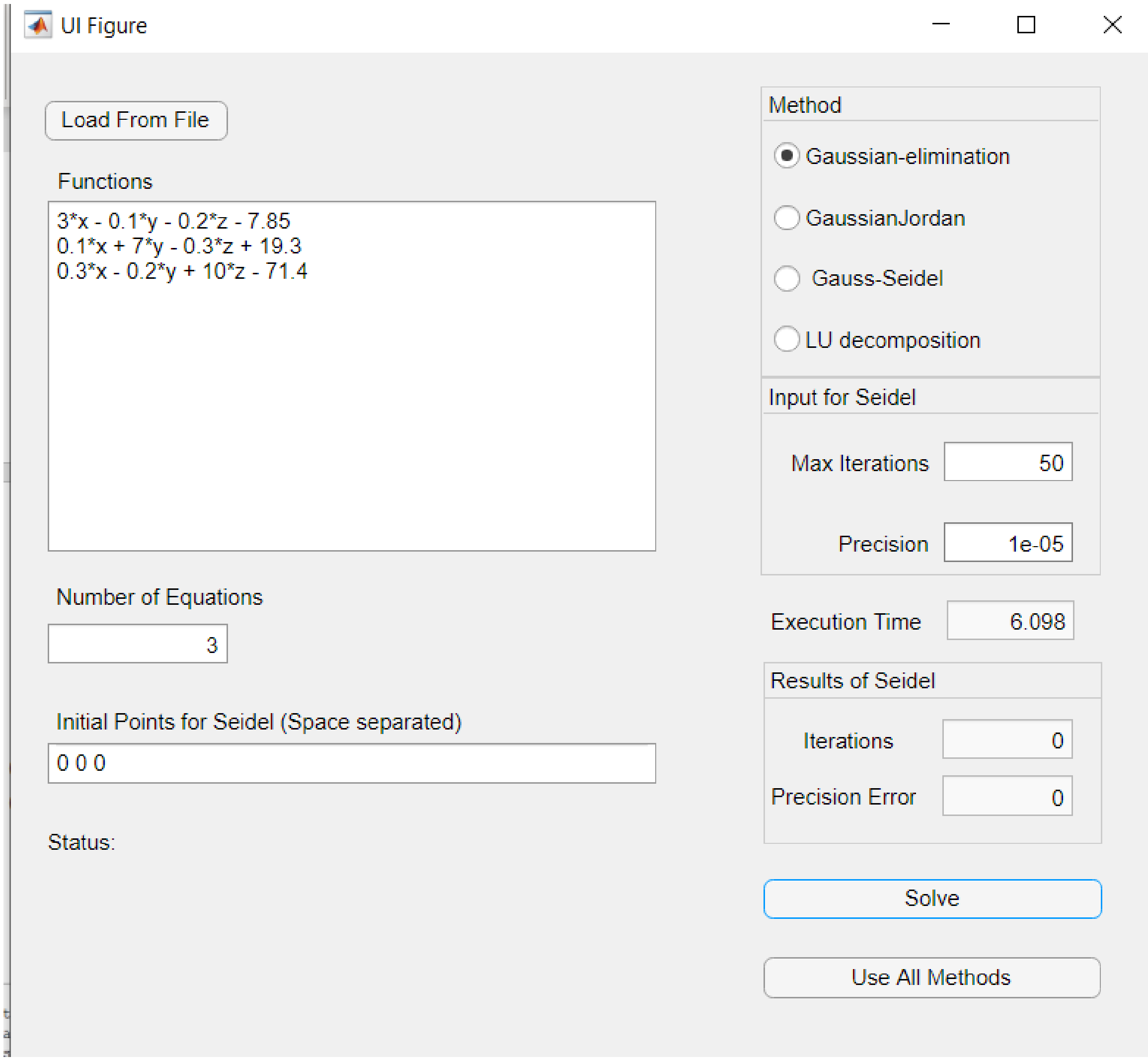
```
169        function xs = gauss_seidel(matrix, no_of_eqs,xs,ea)
170 -          max_iterations = 1;
171 -          while ( max_iterations <= 50 && ea > 0.00001)
172 -              errors = xs;
173 -              for i = 1:no_of_eqs
174 -                  a = matrix(i,i);
175 -                  sum =  matrix(i,no_of_eqs+1);
176 -                  for j = 1:no_of_eqs
177 -                      if (i ~= j)
178 -                          sum = sum + (matrix(i,j) * xs(j))*-1;
179 -                      end
180 -                  end
181 -                  xs(i) = round(sum/a,6);
182 -                  errors(i) = round(abs(((xs(i) -  errors(i))/ xs(i))),2);
183 -              end
184 -              ea = max(errors); max_iterations = max_iterations + 1;
185 -          end
186 -      end
```

- gaussian Seidel is the only iterative method we have

Line 171: while loop to check it the iterations did not exceed the max iteration or the iteration number the user entered and the ea should be less than the precision

- in the above algorithm, I set the max iteration and ea to fixed numbers just to explain
- what we want to do is solve each equation on a different variable
- as we have a matrix so we will get all the elements on the other side and divide them by the coefficient of the current variable
- at the end we will get the max error as the value of ea

The GUI:



Functions

```
3*x - 0.1*y - 0.2*z - 7.85
0.1*x + 7*y - 0.3*z + 19.3
0.3*x - 0.2*y + 10*z - 71.4
```

Number of Equations: 3

Initial Points for Seidel (Space separated): 0 0 0

Status:

Method

● Gaussian-elimination
○ GaussianJordan
○ Gauss-Seidel
○ LU decomposition

Input for Seidel

Max Iterations: 50
Precision: 1e-05

Execution Time: 6.098

Results of Seidel

Iterations: 0
Precision Error: 0

Input:

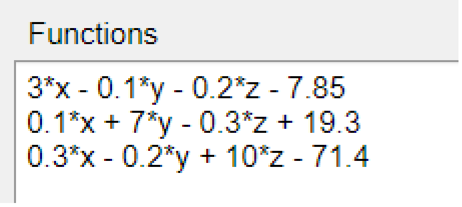
The GUI provides space for the user to enter as many equations in as many variables as they want, and the initial points in case of Gauss-Seidel. The user is required to enter the number of equations they intend to provide. All those fields can be filled automatically from a file using “Load From File” button.

The User choses the solution method from the side panel, and, if they wish, they can choose to solve a given system using all 4 methods by pressing the “Use all Methods” button. This will show them the solutions of each method to the particular input.

For Gauss-seidel (The iterative method) fields for required precision and maximum number of iterations are provided.

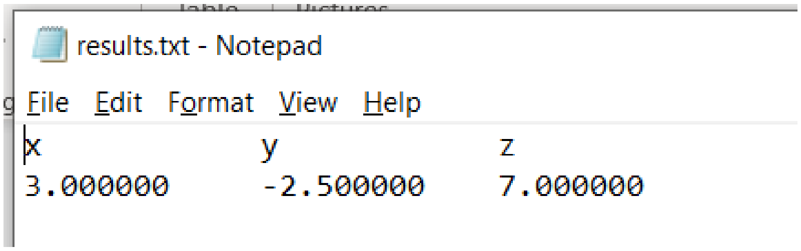
Sample runs:

Input:



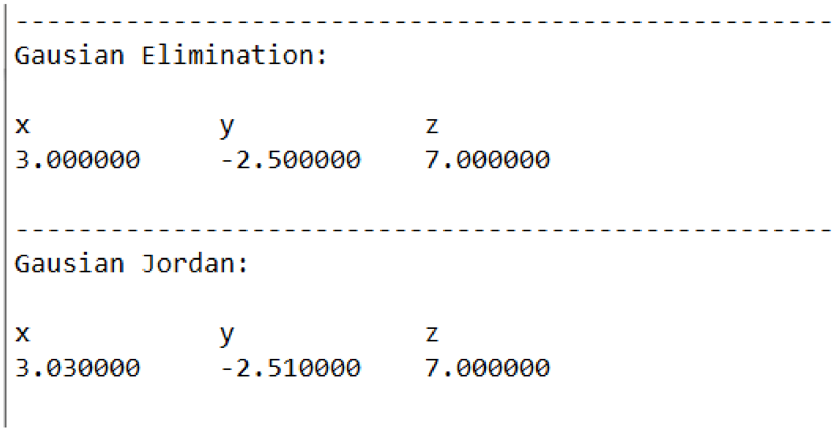
```
3*x - 0.1*y - 0.2*z - 7.85
0.1*x + 7*y - 0.3*z + 19.3
0.3*x - 0.2*y + 10*z - 71.4
```

Result of Gaussian Elimination:

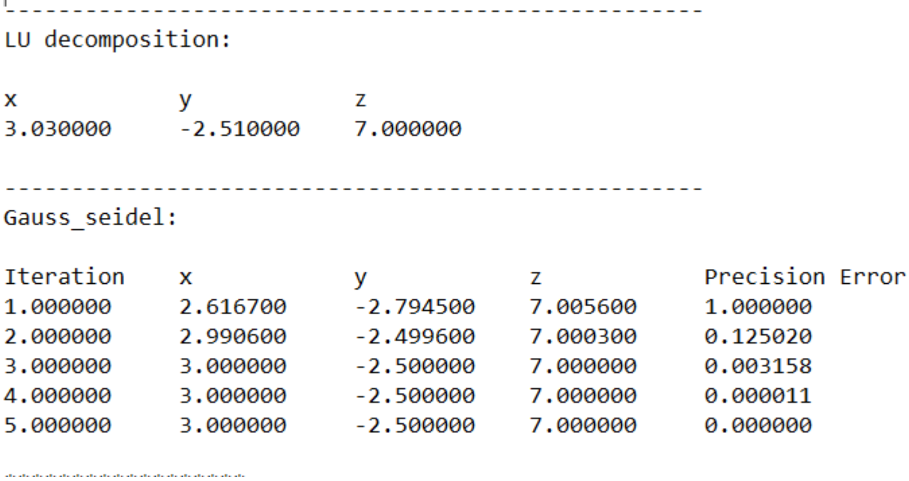


```
x              y              z
3.000000       -2.500000      7.000000
```

Result using all methods:

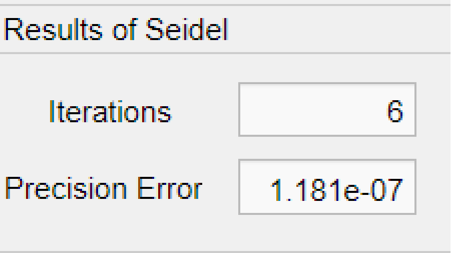


```
---------------------------------------------------
Gausian Elimination:

x            y            z
3.000000     -2.500000    7.000000

---------------------------------------------------
Gausian Jordan:

x            y            z
3.030000     -2.510000    7.000000
```



```
---------------------------------------------------
LU decomposition:

x            y            z
3.030000     -2.510000    7.000000

---------------------------------------------------
Gauss_seidel:

Iteration    x            y            z            Precision Error
1.000000     2.616700     -2.794500    7.005600     1.000000
2.000000     2.990600     -2.499600    7.000300     0.125020
3.000000     3.000000     -2.500000    7.000000     0.003158
4.000000     3.000000     -2.500000    7.000000     0.000011
5.000000     3.000000     -2.500000    7.000000     0.000000
```

Seidel specific results:

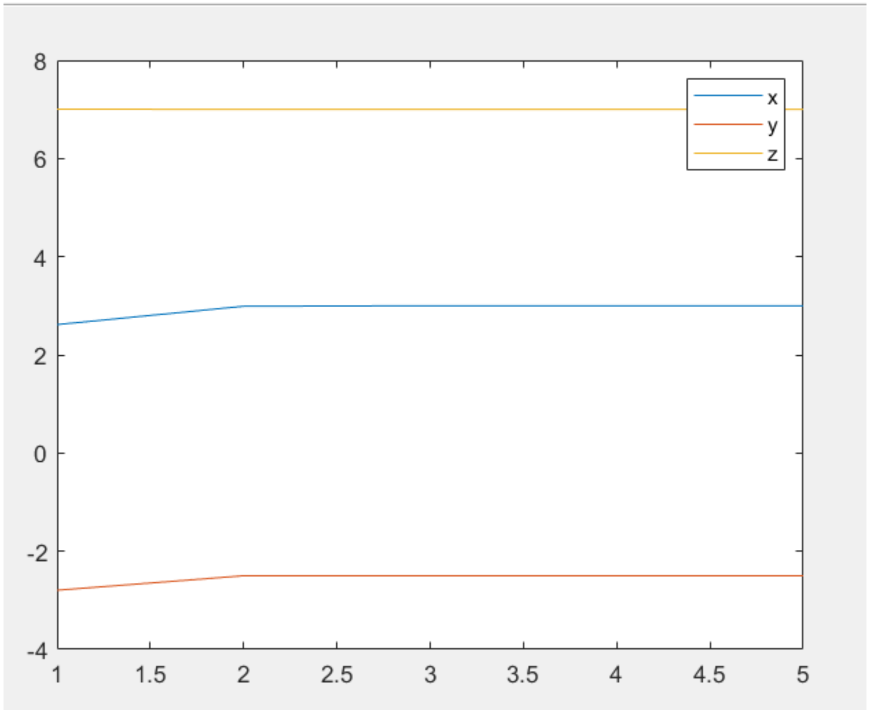


Results of Seidel

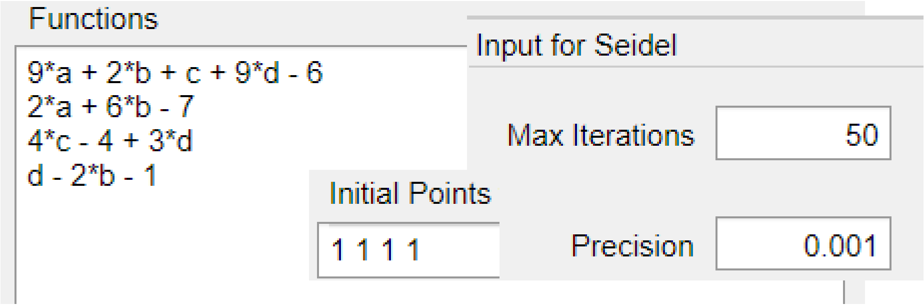
Iterations 6

Precision Error 1.181e-07

Plot of value of variable on each iteration:



Input:



Functions

9*a + 2*b + c + 9*d - 6
2*a + 6*b - 7
4*c - 4 + 3*d
d - 2*b - 1

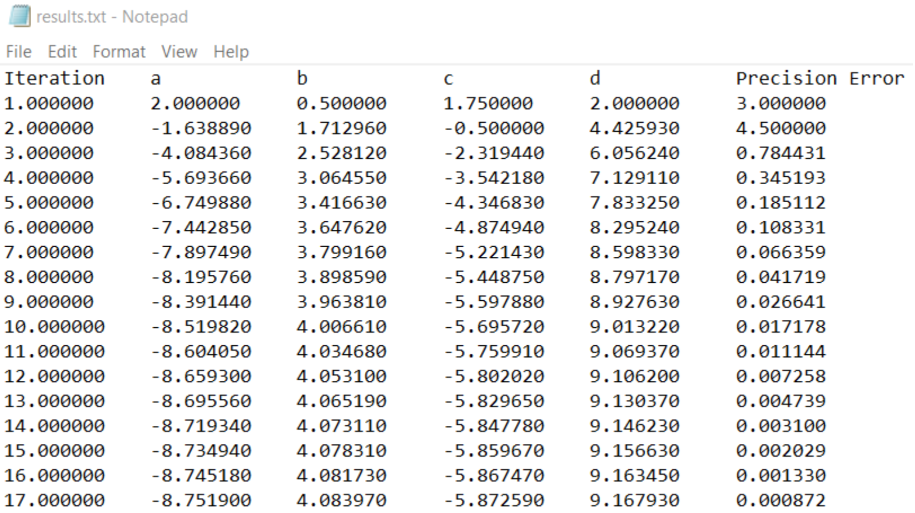
Input for Seidel

Max Iterations: 50

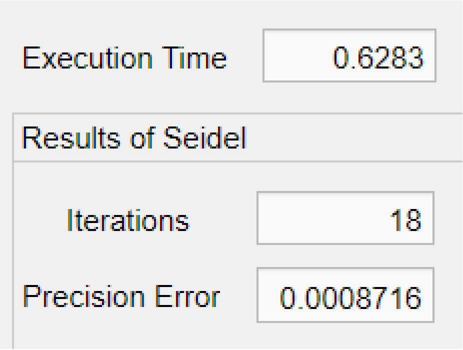
Initial Points: 1 1 1 1

Precision: 0.001

Results using Gauss-seidel:



```
Iteration    a            b           c            d           Precision Error
1.000000     2.000000     0.500000    1.750000     2.000000    3.000000
2.000000     -1.638890    1.712960    -0.500000    4.425930    4.500000
3.000000     -4.084360    2.528120    -2.319440    6.056240    0.784431
4.000000     -5.693660    3.064550    -3.542180    7.129110    0.345193
5.000000     -6.749880    3.416630    -4.346830    7.833250    0.185112
6.000000     -7.442850    3.647620    -4.874940    8.295240    0.108331
7.000000     -7.897490    3.799160    -5.221430    8.598330    0.066359
8.000000     -8.195760    3.898590    -5.448750    8.797170    0.041719
9.000000     -8.391440    3.963810    -5.597880    8.927630    0.026641
10.000000    -8.519820    4.006610    -5.695720    9.013220    0.017178
11.000000    -8.604050    4.034680    -5.759910    9.069370    0.011144
12.000000    -8.659300    4.053100    -5.802020    9.106200    0.007258
13.000000    -8.695560    4.065190    -5.829650    9.130370    0.004739
14.000000    -8.719340    4.073110    -5.847780    9.146230    0.003100
15.000000    -8.734940    4.078310    -5.859670    9.156630    0.002029
16.000000    -8.745180    4.081730    -5.867470    9.163450    0.001330
17.000000    -8.751900    4.083970    -5.872590    9.167930    0.000872
```

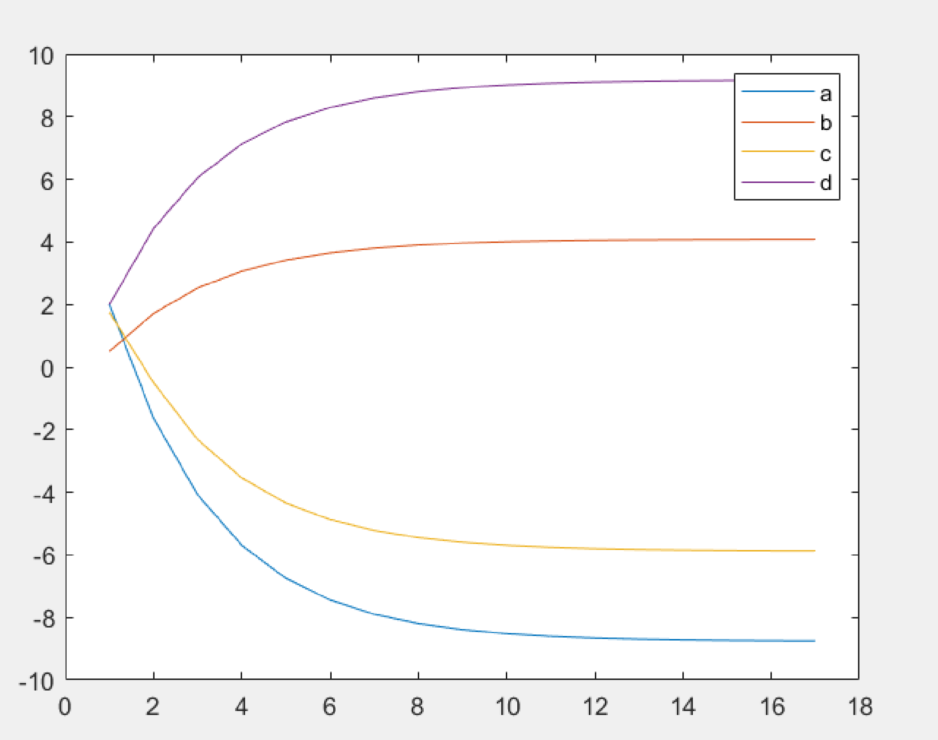


Execution Time: 0.6283

Results of Seidel

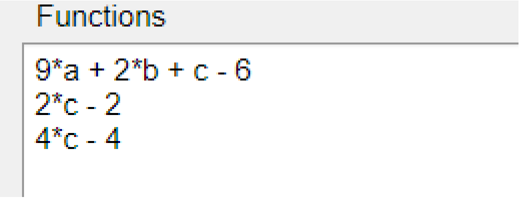
Iterations: 18

Precision Error: 0.0008716



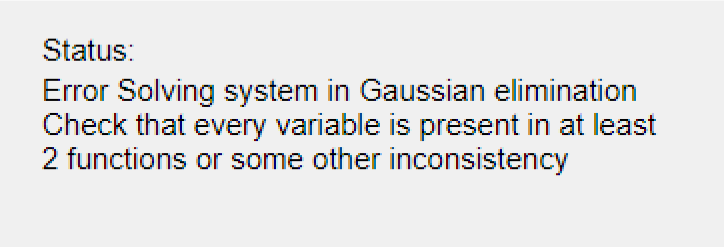
Examples of bad inputs:

- Bad input equations:



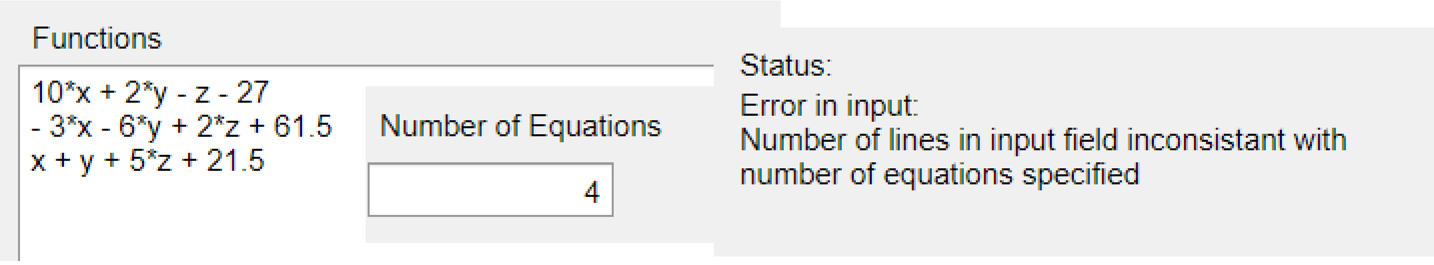
Functions

```
9*a + 2*b + c - 6
2*c - 2
4*c - 4
```



```
Status:
Error Solving system in Gaussian elimination
Check that every variable is present in at least
2 functions or some other inconsistency
```

- Wrong number of equations:



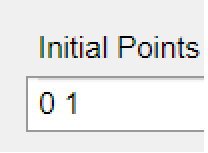
Functions

```
10*x + 2*y - z - 27
- 3*x - 6*y + 2*z + 61.5
x + y + 5*z + 21.5
```

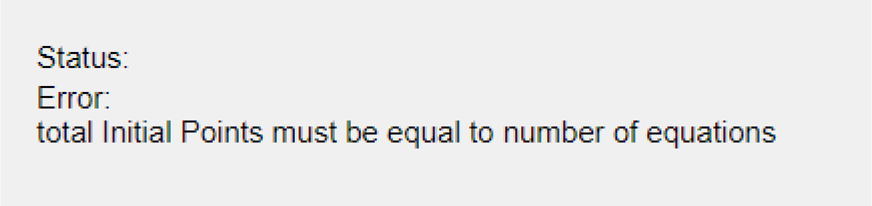
Number of Equations: 4

```
Status:
Error in input:
Number of lines in input field inconsistant with
number of equations specified
```

- Or, in case of Gauss-Seidel for the previous input



Initial Points: 0 1

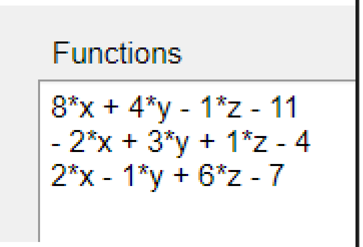


```
Status:
Error:
total Initial Points must be equal to number of equations
```

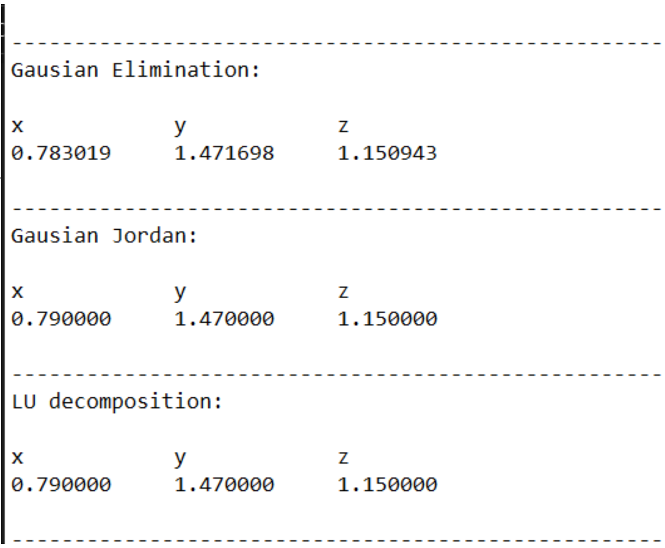
Provided test cases:

NOTE: Results are also provided in txt files accompanying report

Test 1:

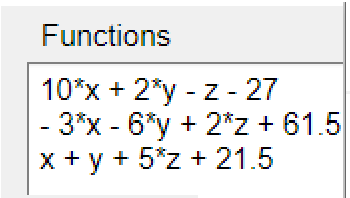


```
Functions
8*x + 4*y - 1*z - 11
- 2*x + 3*y + 1*z - 4
2*x - 1*y + 6*z - 7
```

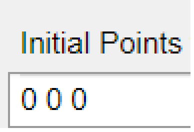


```
-----------------------------------------------------
Gausian Elimination:

x             y             z
0.783019      1.471698      1.150943

-----------------------------------------------------
Gausian Jordan:

x             y             z
0.790000      1.470000      1.150000

-----------------------------------------------------
LU decomposition:

x             y             z
0.790000      1.470000      1.150000

-----------------------------------------------------
```

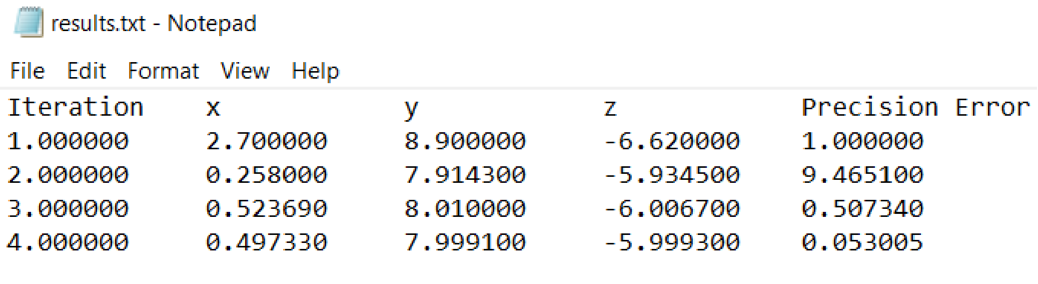
Test 2:



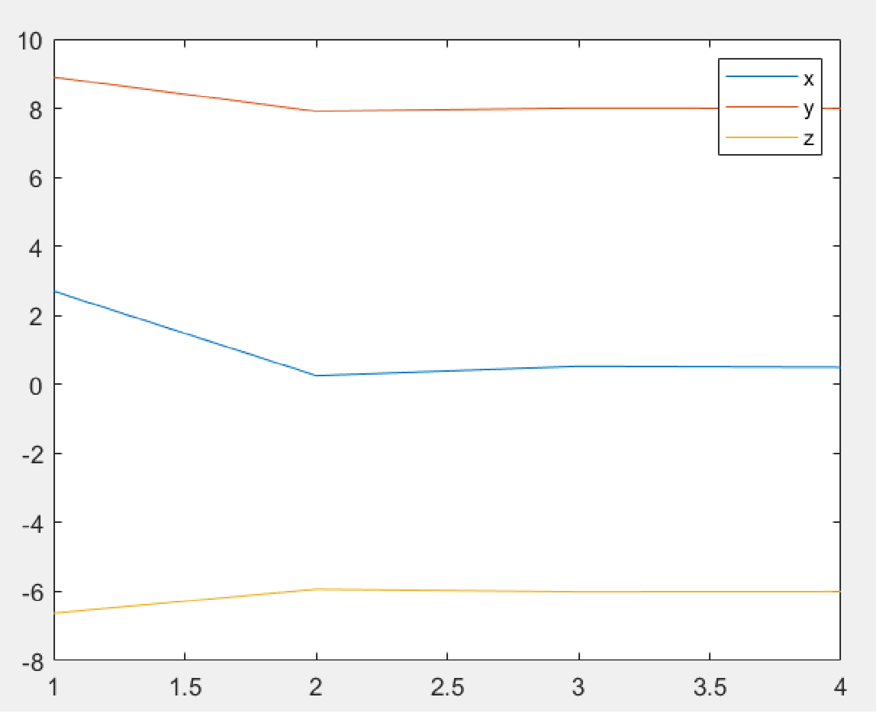
```
Functions
10*x + 2*y - z - 27
- 3*x - 6*y + 2*z + 61.5
x + y + 5*z + 21.5
```



```
Initial Points
0 0 0
```

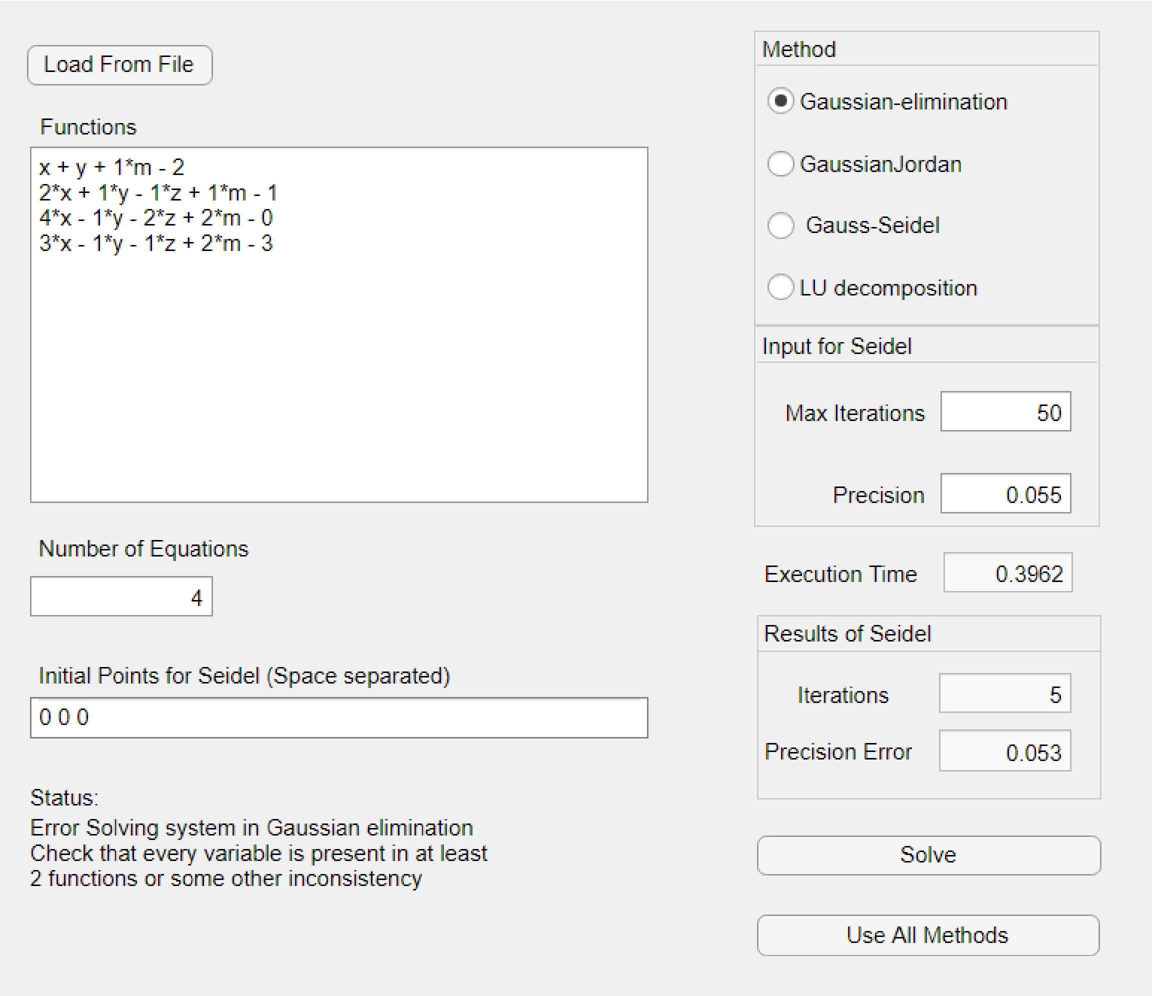


```
Iteration    x           y           z            Precision Error
1.000000     2.700000    8.900000    -6.620000    1.000000
2.000000     0.258000    7.914300    -5.934500    9.465100
3.000000     0.523690    8.010000    -6.006700    0.507340
4.000000     0.497330    7.999100    -5.999300    0.053005
```



Test 3:

It did result in an error, caused by a value that is nan of inf, suggesting a case of division by zero, though the case was not thoroughly investigated.



Functions

```
x + y + 1*m - 2
2*x + 1*y - 1*z + 1*m - 1
4*x - 1*y - 2*z + 2*m - 0
3*x - 1*y - 1*z + 2*m - 3
```

Number of Equations: 4

Initial Points for Seidel (Space separated): 0 0 0

Status:
Error Solving system in Gaussian elimination
Check that every variable is present in at least
2 functions or some other inconsistency

Method

● Gaussian-elimination
○ GaussianJordan
○ Gauss-Seidel
○ LU decomposition

Input for Seidel

Max Iterations: 50
Precision: 0.055

Execution Time: 0.3962

Results of Seidel

Iterations: 5
Precision Error: 0.053