Report Assignment 3 - Numerical Linear Algebra

1- Intermediate results difference for Inverse and In-Order for Jacobi & Gauss-Seidel

I do expect the intermediate results to be different when using Gauss-Seidel, since for a certain iteration 'i', if some value for the approximation x has already being computed, it will use it already in the same iteration. So, for example, if we use a vector 'v' and we are doing in order Gauss-Seidel approximation, that means that in the formula, the components of x already calculated will be used, and as we move wihtin the componenets of the vector 'v', those componenets already approximated in this iteration will already be used. So, for the last component in an in-order approximation, we will already have all the current approximations for all the other componenets, which will hopefully give us better results. The same logic apply for the Inverse, but on the reverse order, meaning that whejn we get to the first component of 'v', we will have already computed all the previous ones for this iteration 'i'.

However, for the Jacobi method it does not matter at all in which order we do the operations, since it will not use any componenets of the new iteration, just of the previous one. Meaning that no matter the order we do, the results will be the same.

To prove this, I have implemented my methods in such a way that one of the parameters allows to choose for In-Order or Inverse approximation, both for Jacobi and Gauss-Seidel.

2- Which Stopping Criteria did you use for your Implementation?

I have decided to use as a stoping crtieria the infinity norn of the difference between the current approximation and the previous one.

The reason why I decided to use the Infinity norm is that, since it basically returns the absolute value of the biggest component of the vector , I am certain that for all other components, my approximation is bellow the error inputed by the user.

Another option would be to use the two-norm (eucledian lenght) and compare that to the error. However, since the infinity norm is always smaller than or equal to the two-norm, it could be the case that all the individial elements of the vector are already bellow the error entered by the user, but we do a few more iterations due to the value of the two-norm. Therefore, I have prefered to use the Infinity, since we can literally stop the moment that each individual component has reached a value bellow the error.

Or, since we are using Matlab, we could have compared our current approximation to the exact value (though for a real life scenario or for a very large matrix, this option would be not so useful).

3- How many iterations do both methods need for d = {1, 2, 3, 4, 5}? Do both methods converge? Why (not)? Why is d = 4 a special case?

First of all, I beleive that the question meant to say why d = 5 is a special case, because for all the other values of d, besides 5, both of my methods converged.

However, for d = 5, my approximation did not converge a numerical value, but to this infinity vector as seen in the picture bellow. One reason that could explain this weird behaviour for d = 5 is the fact that on A5, the second and third rows are not diagonally dominant, which means that there is already not a guarantee of convergence.

```
x_jacobi_A5 =

   Inf
   Inf
  -Inf


n_iterations_jacobi_A5 =

      106430


x_gauss_seidel_A5 =

   Inf
   Inf
  -Inf


n_iterations_gauss_seidel_A5 =

       46289


x_exact_A5 =

        -10.3333333333334
        -19.0000000000001
         15.6666666666667
```

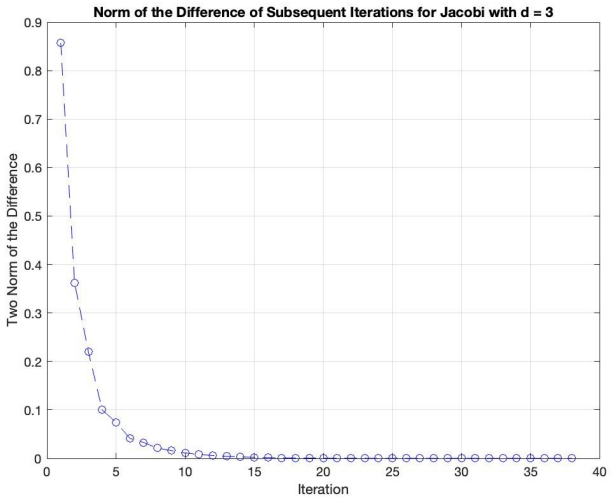
I tried to see a reason why this exercise would ask for the value of d = 4 and, two possible reaons is that, first of all, the Matrix A3 is not diagonally dominant, which means that congergence using an iterative method is not guaranteed (though it converged in this case). On the second row, |5| is not greater ot equal to $|-2| + |4|$. The second reason would be the fact that A4 took significantly more iterations to converge when compare to the other values of d (excluding 5, of course).

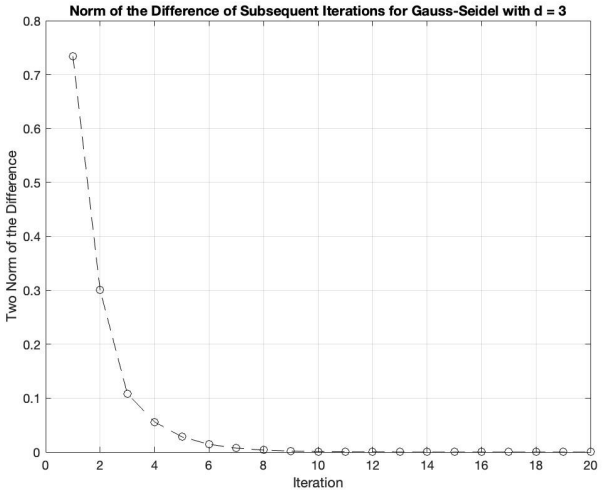
```
x_jacobi_A4 =

         0.317754033452586
         1.45793805418773
        -0.663546533459113


n_iterations_jacobi_A4 =

    84


x_gauss_seidel_A4 =

         0.317755943293996
         1.45794155212026
        -0.663549167752918


n_iterations_gauss_seidel_A4 =

    41


x_exact_A4 =

         0.317757009345795
         1.45794392523365
        -0.663551401869159
```

The second highest number of iterations for Jacobi had been A3 with 38. And, the second highest foir Gauss-Seidel has been 20, also with A3.

4- Plotting the two-norm of the difference between subsequent iterations for d = 3. What do you see? Why does this occur?

Here are the plots:





I see a very interesting shape that looks a lot like an exponential decay function for both methods. This is a very powerful insight, because it gives us an idea of how these methods converge to a certain error given a certain number of iterations. This also explains visually how fast both methods converged to the solutions of A1, A2, and A3, even though the error was of 10^-6.

Besides, we see that though the shape is the same, Gauss-Seidel converges much quicker than Jacobi and the inital norm for the error is already smaller.

One intuitive way to explain this phenemena is that at each iteration, we are "chopping the problem", meaning that not only we are getting closer, but getting closer much faster. Simular to the idea of binary search, that in each iteration, we can "chop half of the problem away" (though of course, for this is not necessaraly half, but still an exponential decay)

5- Would you prefer to use Jacobi or Gauss-Seidel method to solve linear systems iteratively? Why?

I would defintely prefer to use Gauss-Seidel to solve problems involving linear systems. Without any exceptions (for the systems that converged), for all values of 'd', the number of iterations was considerably smaller using Gauss-Seidel when compared to Jacobi.

```
n_iterations_jacobi_A1 =
    16

n_iterations_gauss_seidel_A1 =
     9

n_iterations_jacobi_A2 =
    23

n_iterations_gauss_seidel_A2 =
    13

n_iterations_jacobi_A3 =
    38

n_iterations_gauss_seidel_A3 =
    20
```

```
n_iterations_jacobi_A4 =
    84

n_iterations_gauss_seidel_A4 =
    41
```

Besides, if you think about it, we are not computting anything extra by using Gauss-Seidel, but we are simply updating the new approximation's components as we find then, instead of waiting for the whole iteration to finish to just then start using those values.

Results of the Code

```
x_jacobi_A1 =

        -0.190184174733433
         0.754600958573909
        -0.153373918679416


n_iterations_jacobi_A1 =

    16


x_gauss_seidel_A1 =

        -0.190184082730014
         0.754601112329348
        -0.153373977640499


n_iterations_gauss_seidel_A1 =

     9


x_exact_A1 =

        -0.190184049079755
         0.754601226993865
        -0.153374233128834
```

```
x_jacobi_A2 =

        -0.104869471981884
         0.831459485217149
         -0.18351943788618


n_iterations_jacobi_A2 =

    23


x_gauss_seidel_A2 =

        -0.104868991467169
         0.831460433289007
        -0.183520227283125


n_iterations_gauss_seidel_A2 =

    13


x_exact_A2 =

        -0.104868913857678
         0.831460674157303
        -0.183520599250936
```

```
x_jacobi_A3 =

        0.0204069725881831
         0.979589394652421
        -0.285712134213813


n_iterations_jacobi_A3 =

    38


x_gauss_seidel_A3 =

        0.0204079096813093
         0.979591167424061
        -0.285713504222219


n_iterations_gauss_seidel_A3 =

    20


x_exact_A3 =

        0.0204081632653061
         0.979591836734694
        -0.285714285714286
```

```
x_jacobi_A4 =

         0.317754033452586
          1.45793805418773
        -0.663546533459113


n_iterations_jacobi_A4 =

    84


x_gauss_seidel_A4 =

         0.317755943293996
          1.45794155212026
        -0.663549167752918


n_iterations_gauss_seidel_A4 =

    41


x_exact_A4 =

         0.317757009345795
          1.45794392523365
        -0.663551401869159
```

```
x_jacobi_A5 =

   Inf
   Inf
  -Inf


n_iterations_jacobi_A5 =

      106430


x_gauss_seidel_A5 =

   Inf
   Inf
  -Inf


n_iterations_gauss_seidel_A5 =

       46289


x_exact_A5 =

         -10.3333333333334
         -19.0000000000001
          15.6666666666667
```