LAB 4 - ITERATIVE METHODS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS

1. Jacobi's method

Consider the following system of two linear equations in two variables:

$$\begin{aligned} 7x - y &= 6 \\ x - 5y &= -4 \end{aligned} \tag{1}$$

Problem 1. Solve the above system by hand, and save the x value in a variable called `x_val` and the y value in a variable called `y_val`.

Instead of solving the system (1) directly, we could instead start with an initial starting estimate, say $(x_0, y_0) = (0, 0)$, and then use this estimate to hopefully create better and better approximations to the actual solution (`x_val`, `y_val`) of the system. To see how this is done, solve the first equation in the above system for x, and the second equation for y. We obtain the following pair of equations:

$$x = \tfrac{1}{7}(6 + y) \tag{2}$$

$$y = \tfrac{1}{5}(x + 4). \tag{3}$$

By plugging our initial estimates $x_0 = 0$ and $y_0 = 0$ into these equations, we obtain a second improved estimate, which we call (x_1, y_1). More precisely, we plug the value $y_0 = 0$ into equation (2) to get x_1, and we plug the value of x_0 into equation (3) to get y_1:

$$\begin{aligned} x_1 &= \tfrac{1}{7}(6 + y_0) = \tfrac{1}{7}(6 + 0) = \tfrac{6}{7} \\ y_1 &= \tfrac{1}{5}(x_0 + 4) = \tfrac{1}{5}(0 + 4) = \tfrac{4}{5} \end{aligned}$$

- Verify that the estimate (x_1, y_1) is actually an improvement over $(x_0, y_0) = (0, 0)$ by comparing it to the values of (`x_val`,`y_val`) you found earlier.

Since the values we obtained by plugging (x_0, y_0) into equations (2) and (3) seemed to improve our estimate, it's reasonable to think we might improve our estimates even further by plugging $(x_1, y_1) = (\frac{6}{7}, \frac{4}{5})$ into equations (2) and (3). Indeed, doing this gives us a new improved estimate (x_2, y_2) as follows

$$\begin{aligned} x_2 &= \tfrac{1}{7}(6 + y_1) = \tfrac{1}{7}\left(6 + \tfrac{6}{7}\right) = \tfrac{48}{49} \\ y_2 &= \tfrac{1}{5}(x_1 + 4) = \tfrac{1}{5}\left(\tfrac{4}{5} + 4\right) = \tfrac{24}{25}. \end{aligned}$$

You can see that our approximations seems to be improving each time we plug the preceding values into our equations, so we might as well continue on. We can define a sequence of estimated solutions (x_n, y_n) by iterating (or repeating) the above steps. Once we've computed

(x_{n-1}, y_{n-1}), we plug the value y_{n-1} into equation (2) to get x_n, and we plug the value x_{n-1} into equation (3) to get y_n.

We hope that the sequence of values (x_n, y_n) will converge to the actual solution of the system (i.e., we hope that the values of (x_n, y_n) will get closer and closer to (`x_val`, `y_val`) as n gets larger). This procedure for solving a system of linear equations is known as *Jacobi's method*, and is an example of an iterative method for solving a system of linear equations.

In the following two problems we will write functions which return the values of x_n and y_n for system (1) above, using Jacobi's method. We will use the starting approximation $(x_0, y_0) = (0, 0)$ like we did above.

Problem 2. Define a function, called `jacobi1_iteration(x,y)`, which accepts as input values `x` and `y` (which we think of as being x_{n-1} and y_{n-1} respectively), and returns a list `[new_x,new_y]`, where `new_x` is the updated value x_n and `new_y` is the updated value y_n using Jacobi's method and equations (2) and (3).

In other words, `jacobi1_iteration(x,y)` should return the results of performing one iteration of Jacobi's method for system (1) on the inputs `x`, `y`.

To test your function, note that `jacobi1_iteration(3,5)` should return `[1.5714285714285714, 1.4]`.

Recall that because of roundoff errors, the answers you get might not match the ones above exactly. If your answer matches to 15 or 16 decimal points then you have probably coded your function correctly.

Problem 3. Define a function, called `jacobi1_method(n)` which accepts as input a single non-negative integer `n`, and returns a list `[x_n,y_n]`, where `x_n` and `y_n` are the values of x_n and y_n respectively for Jacobi's method when applied to system (1) above. Use $(x_0, y_0) = (0, 0)$ as your starting approximation.

Hint: Your function `jacobi1_method` *can call the function* `jacobi1_iteration` *that you defined in the previous problem.*

To test your function, note that `jacobi1_method(3)` should return the list `[0.9959183673469387,0.9942857142857143]`.

Problem 4. Use the function `jacobi1_method` to answer the questions below.

(1) What is the smallest value of n so that x_n and y_n are both within 0.1 of the values for `x_val` and `y_val` respectively? Save the value of the smallest such n as a variable called `n_var1`.
(2) What is the smallest value of n so that x_n and y_n are both within 0.0001 of the values for `x_val` and `y_val` respectively? Save the value of the smallest such n as a variable called `n_var2`.

2. Gauss-Seidel method

We can change the above procedure to obtain a different method for solving systems of linear equations, called the *Gauss-Seidel method.* The only difference between these two methods is the timing of when we plug in the values of x_{n-1} and y_{n-1} to get x_n and y_n. In the Gauss-Seidel method we compute x_n by plugging y_{n-1} into equation (2), the same as we do in Jacobi's method. However, when finding y_n we plug the newly computed value of x_n into equation (3), instead of plugging in x_{n-1} as we would do in Jacobi's method. In other words, we always plug in the most recently computed values for x and y when computing x_n and y_n.

In the above example, starting with $(x_0, y_0) = (0, 0)$ as before, we would compute

$$x_1 = \tfrac{1}{7}(6 + y_0) = \tfrac{1}{7}(6 + 0) = \tfrac{6}{7}$$

and

$$y_1 = \tfrac{1}{5}(x_1 + 4) = \tfrac{1}{5}\left(\tfrac{6}{7} + 4\right) = \tfrac{34}{35}.$$

Problem 5. Define a function, called `gs1_iteration(x,y)`, which accepts as input values for `x` and `y` (which we think of as being x_{n-1} and y_{n-1} respectively), and returns a list `[new_x,new_y]`, where `new_x` is the updated value x_n and `new_y` is the updated value y_n using the Gauss-Seidel method and equations (2) and (3).

In other words, `gs1_iteration(x,y)` should return the results of performing one iteration of the Gauss-Seidel method for system (1) on the inputs `x`, `y`.

To test your function, note that `gs1_iteration(3,5)` should return `[1.5714285714285714, 1.1142857142857143]`.

Problem 6. Define a function, called `gs1_method(n)`, which accepts as input a single non-negative integer `n`, and returns a list `[x_n,y_n]`, where `x_n` and `y_n` are the values of x_n and y_n respectively for the Gauss-Seidel method when applied to system (1) above. Use $(x_0, y_0) = (0, 0)$ as your starting approximation.

To test your function, note that `gs1_method(2)` should return the list `[0.9959183673469388, 0.9991836734693879]`.

Problem 7. Use the function `gs1_method(n)` to answer the questions below.

(1) What is the smallest value of n so that x_n and y_n are both within 0.1 of the values for `x_val` and `y_val` respectively? Save the value of the smallest such n as a variable called `n_var3`.

(2) What is the smallest value of n so that x_n and y_n are both within 0.0001 of the values for `x_val` and `y_val` respectively? Save the value of the smallest such n as a variable called `n_var4`.

- Which of the two methods for solving this system seems to be converging faster?

3. Error

For every n, we can think of the values (x_n, y_n), which come either from Jacobi's method or the Gauss-Seidel method, as being an approximation to the solution of system (1). We can therefore measure how close our approximation is by computing the distance from the point (x_n, y_n) to the point (`x_val`,`y_val`).

If `a` and `b` are two NumPy arrays of the same size then we can compute the distance between `a` and `b` by the command

```
np.linalg.norm(a-b)  # a and b must be NumPy arrays here, and not lists.
```

Remember that you will need to import NumPy in this notebook before you can use the function `np.linalg.norm` or any of the other NumPy functions in this lab. Another thing you may need to remember is how to convert lists to arrays. For example, if you have two lists `list1` and `list2` and you want to find the distance between them (thinking of them as vectors), you will need to cast them as NumPy arrays when plugging them into the `np.linalg.norm` function as follows:

```
np.linalg.norm(np.array(list1)-np.array(list2))
```

As you probably guessed, when we plug the list `list1` into the function `np.array` it returns a NumPy array with the same values as `list1`. In other words `np.array` takes lists and converts them into arrays.

Problem 8. Define a function `gs1_error(n)` that accepts as input a non-negative integer `n` and outputs the error of the Gauss-Seidel approximation (x_n, y_n) to the solution of system (1). In other words, it should output the distance between the nth approximation (x_n, y_n) and the true value (`x_val`,`y_val`) of the solution to (1).

To test your code, the output for `gs1_error(3)` should be `0.0001189275688300568`.

Create a graph that plots the error of the estimate (x_n, y_n) for the first 50 values of n (i.e. for $n = 0, 1, \ldots, 49$). We will provide you with the code you need to create the plot, but you should read through it to make sure you understand what each step is doing (you should be familiar with most of the commands).

4. Convergence of the Jacobi and Gauss-Seidel methods

A question we should be asking at this point is whether or not the Jacobi and Gauss-Seidel methods always converge, and when they do converge, do they always converge to a solution of the system? The answer to the second question is "yes," i.e. whenever the estimated values (x_n, y_n) converge to some values (x', y') as n gets large, it is guaranteed that these (x', y') values are a solution to the system.

The problem is that the sequence of estimates (x_n, y_n) doesn't neccessarily have to converge to any values at all. Consider the following system:

$$\begin{aligned} x - y &= 1 \\ 2x + y &= 5 \end{aligned} \tag{4}$$

Problem 9. Define functions `gs2_iteration(x,y)` and `gs2_method(n)` as above, this time to produce the estimates (x_n, y_n) as computed by the Gauss-Seidel method applied to system (4). Use $(x_0, y_0) = (0, 0)$ again as the starting estimate.

Important: To simplify the grading, solve the first equation for x, and the second equation for y. This isn't necessary to find the solution, but in order to have your work graded you will need to solve the equations in this order.

Define an error function `gs2_error(n)` as above, and create a plot that shows the error of the Gauss-Seidel estimates for the first 50 values of n.

To test your code, note that `gs2_method(5)` should return the list `[-14, 33]`, and `gs2_error(3)` should return `8.94427190999916`.

- What do you notice when you compare the plots `gs1_error` and `gs2_error`? What does this tell you about the effectiveness of using the Gauss-Seidel method to solve these two systems of equations?

There is a class of matrices for which these methods are guaranteed to converge. An $n \times n$ matrix

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

is said to be *strictly diagonally dominant* if

$$\begin{aligned} |a_{11}| &> |a_{12}| + |a_{13}| + \cdots + |a_{1n}| \\ |a_{22}| &> |a_{21}| + |a_{23}| + \cdots + |a_{2n}| \\ &\vdots \\ |a_{nn}| &> |a_{n1}| + |a_{n2}| + \cdots + |a_{n(n-1)}|. \end{aligned}$$

In other words, an $n \times n$ matrix is strictly diagonally dominant if in each row the absolute value of the term on the diagonal is larger than the sum of the absolute values of the other terms in the row.

Theorem 1. *If a system of n linear equations with n variables has a coefficient matrix which is strictly diagonally dominant, then the system has a unique solution and the Jacobi and Gauss-Seidel methods both converge to it.*

Note that just because the coefficient matrix of a system isn't strictly diagonally dominant doesn't mean these methods won't converge. In that case the methods might converge or they might diverge.

- Look back at the previous systems of equations we have studied in this lab. For which of these systems does Theorem 1 apply? What does this tell you about whether or not we can expect the Jacobi and Gauss-Seidel methods to converge for these systems?

Problem 10. Similar to the problems above, define functions called `gs3_iteration(x,y,z)` and `gs3_method(n)` which use the Gauss-Seidel method and starting approximation

$$(x_0, y_0, z_0) = (0, 0, 0)$$

to solve the following system

$$\begin{aligned} 5x - 2y + 3z &= -8 \\ x + 4y - 4z &= 102 \\ -2x - 2y + 4z &= -90 \end{aligned}$$

Important: When writing your code solve the first equation for x, the second equation for y, and the third equation for z. This is again to simplify the grading of your lab work.

To test your code, note that `gs3_method(4)` should return the list `[10.74020625, 11.6154796875, -11.32215703125]`.

- Does the Gauss-Seidel method applied to this system seem to converge or diverge? Do your observations contradict the above theorem?

In closing, you may be wondering why we bother to use iterative methods like Jacobi's method or the Gauss-Seidel method, when they aren't guaranteed to converge, and there are perfectly good direct ways to solve these systems (e.g. row reduction). While it is indeed true that for small systems (i.e. a few equations in a few variables) we are probably better off solving them directly, many of the systems that are of interest in the real world are much, much larger, often including tens of thousands of equations in as many variables or more. In cases like these direct methods such as row reduction simply can't be carried out in a reasonable amount of time, while iterative methods can often produce suitable approximations much, much faster. As we will see later on, iterative algorithms provide practical approaches to solving many problems in linear algebra and other fields, and form a crucial set of tools for applying mathematics to the real world.