**Slide 1 (0:00)**

Good morning! My name is Ivan Sergeyev. I am a student and I my job is to find solutions to problems. In natural sciences, the problems are solved with the use of equations, most commonly, systems of linear algebraic equations. Numerical methods help us solve equations efficiently. Today I would like to talk about how these algorithms work and how efficient they are, compared to each other.

**Slide 2 (0:25)**

In my talk, I will go over six points. I will begin with stating the original problem and describing the methods we can use to solve it. Then I will go over the structure of the program I have created to compare the algorithms and the results I got from it. By the end of my presentation, you will know how to approach solving equations and what methods are the most efficient.

**Slide 3 (0:52)**

To start, I will briefly talk about the problem at hand – finding the solution of a system of linear algebraic equations. A system of linear algebraic equations is a collection of several linear equations, which consist of terms that are either a constant or a product of a constant and a first power of a single variable. The solution of the system is a set of values of variables such that all equations of the system are simultaneously satisfied.

Commonly, a shorter notation is used involving matrices and vectors. The coefficients of the equations constitute a matrix while the variables and the constant terms constitute vectors. The solution is a vector of variables that satisfies the matrix equation.

Hereinafter *N* or *n* will denote the number of equations in the system, *A* will denote the matrix of the system’s coefficients, *f* will denote the vector of constant terms in the right hand side and *u* will denote the vector of variables.

**Slide 4 (1:52)**

Now that the problem is stated, let us devise an iterative process to solve the system. Given the original matrix equation, let us multiply it by a real constant *τ* and add the vector of variables *u*. Rearranging the terms, we arrive at this equation, equivalent to the original one. Now we can introduce new symbols for the matrix and the vector involved in the equation, denoting them *B* and *F* respectively. We can now introduce an iterative process. On each step, it uses the current vector of variables to calculate the next vector that is a better approximation of the solution of the original system.

**Slide 5 (2:32)**

Having discussed the general idea of iterative approximation, let us move on to specific algorithms that utilize this approach. On this slide three algorithms are presented: the Jacobi method, the Gauss-Seidel method and the Over-relaxation method. Each method is defined by a recalculation rule used on each step. As you can see, these three algorithms are very similar. The Jacobi method performs its iteration exactly as it was described on the previous slide. The Gauss-Seidel method considers each variable in turn during recalculation. To derive the new value of current variable the new values of variables previously recalculated are used. The Over-relaxation method possesses a parameter *τ* that can have a value between 0 and 2. The Over-relaxation method is identical to the Gauss-Seidel method when *τ* is equal to 1.

**Slide 6 (3:21)**

Now that we have described the methods, let me tell you about the structure of the program I have created to measure and compare the performance of the algorithms. In short, there are four steps to it. Firstly, the wrapper objects that containing methods are created. Secondly, each of the algorithms is run on randomly generated tests. The elapsed time is measured and saved in data files. Thirdly, a file is created with plot instructions. Lastly, the program runs an external utility called gnuplot to create a graph containing all the gathered data using the instructions created prior.

**Slide 7 (3:58)**

Knowing how the program works, let us move on to the results. Here you can see a graph of elapsed time in seconds versus the size of the system of equations. The gathered data is depicted with points with error bars. The dependency was fitted with a quadratic polynomial individually for each of the algorithms. The polynomial coefficients are in the top left corner of the graph just below the legend. In this scale, we can see that all algorithms performed similarly, but we can not compare them. Let me zoom in on this part of the graph. I will say a quote from the movies: “Enhance!” Now you can easily compare the performance of all the algorithms.

**Slide 8 (4:42)**

Having seen the graph, let us move on to results. The fastest algorithm was the Over-relaxation method with *τ* equal to 1.5. The same method with *τ* equal to 0.5 was only 0.1% slower. The Gauss-Seidel method came in third place being 0.5% slower. The Jacobi method is 3.1% slower than the fastest algorithm and hence bottoms the list.

**Slide 9 (5:04)**

Having seen the results, let us discuss the improvements that can be made to the program in the future. Firstly, the program is written in Python. Its development did not take much time and the code is simple. The downside is the program’s performance. It took 5 hours to gather all the data. To improve the speed of data taking, the program should be implemented in C or C++. Secondly, the program has become big already and it will become even bigger if it will be rewritten in C. That is why various measures should be taken for the sake of maintenance. Lastly, only 4 algorithms were compared. More algorithms could be added for more thorough research.

**Conclusion (5:47)**

This is the end of my presentation. I hope that something from what I have told you today will be useful for you in the future. Thank you for your attention. Now I will answer your questions if you have any.

**Le Fin (6:02)**