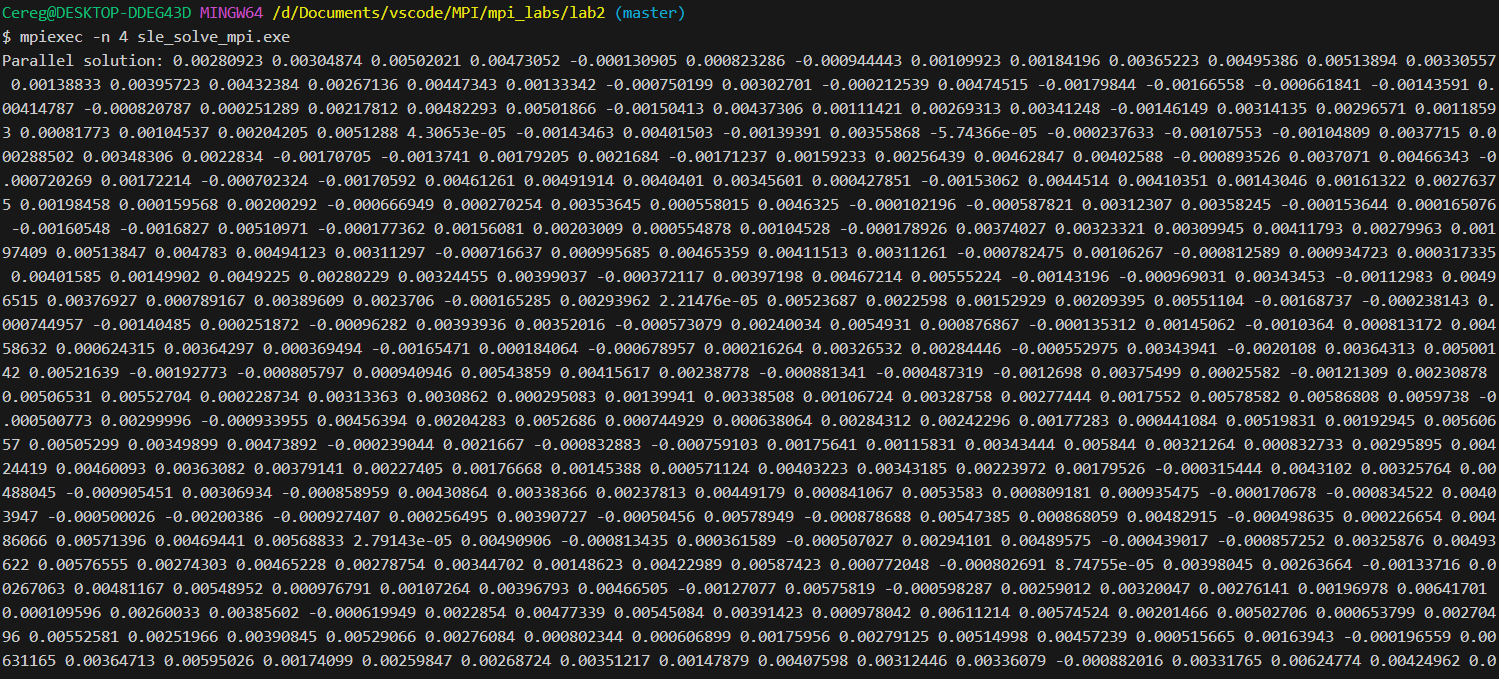
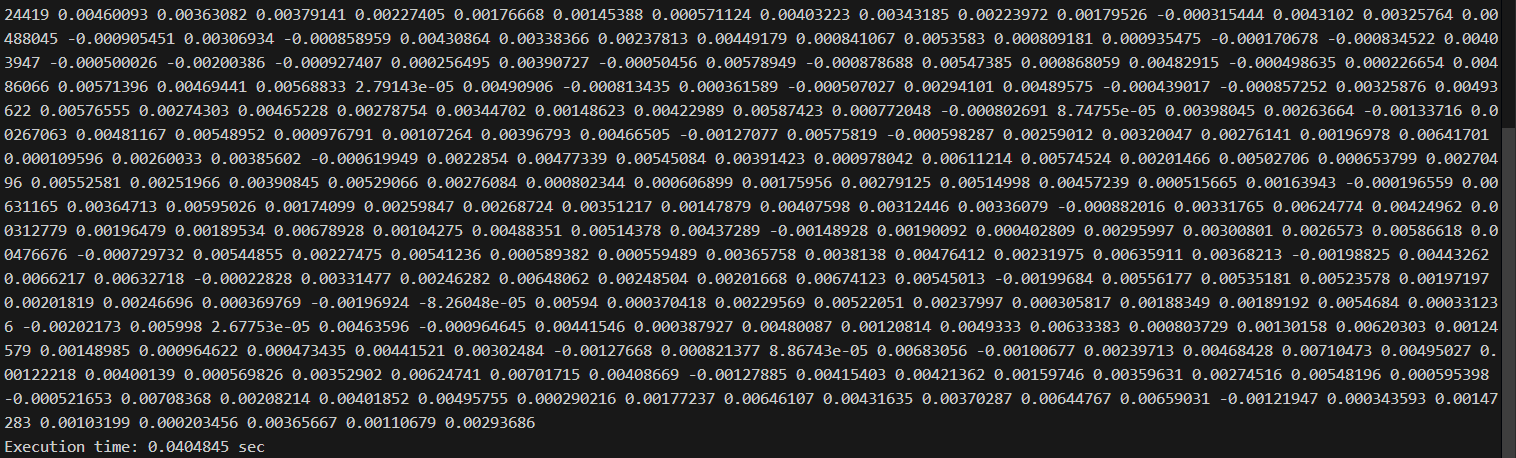
2. «Решение уравнений методом Гаусса»

Принцип лабораторной работы аналогичен первой: составление и проверка алгоритма решения уравнений методом Гаусса на примерах с малым количеством уравнений (2-3), сравнение времени выполнения программы в разных режимах на примере с большим количеством уравнений (от 500).





1. /\*
2. \* Программа решает системы линейных уравнений методом Гаусса.
3. \* Поддерживает два режима работы: последовательный и параллельный (с использованием MPI).
4. \* Особенности:
5. \* - Генерация диагонально доминирующих матриц для устойчивости решения
6. \* - Оптимальное распределение вычислений между процессами в параллельном режиме
7. \* - Замер времени выполнения для сравнения производительности
8. \*/
9. #**include** <mpi.h>
10. #**include** <iostream>
11. #**include** <vector>
12. #**include** <cstdlib>
13. #**include** <ctime>
14. #**include** <cmath>
15. **using** **namespace** std;
16. /\*\*
17. \* Выводит матрицу системы и вектор правых частей
18. \* @param A - матрица коэффициентов
19. \* @param b - вектор правых частей
20. \*/
21. void **print\_matrix**(const vector<vector<double>> &A, const vector<double> &b)
22. {
23. int n = A.size();
24. **for** (int i = 0; i < n; i++)
25. {
26. **for** (int j = 0; j < n; j++)
27. cout << A[i][j] << "\t";
28. cout << "| " << b[i] << endl;
29. }
30. cout << endl;
31. }
32. /\*\*
33. \* Последовательная реализация метода Гаусса с выбором ведущего элемента
34. \* @param A - матрица коэффициентов (модифицируется в процессе)
35. \* @param b - вектор правых частей (модифицируется в процессе)
36. \*/
37. void **sequential\_gaussian\_elimination**(vector<vector<double>> &A, vector<double> &b)
38. {
39. int n = A.size();
40. // Прямой ход метода Гаусса
41. **for** (int i = 0; i < n; i++)
42. {
43. // Поиск максимального элемента в текущем столбце
44. int maxRow = i;
45. **for** (int k = i + 1; k < n; k++)
46. **if** (abs(A[k][i]) > abs(A[maxRow][i]))
47. maxRow = k;
48. // Обмен строк для улучшения устойчивости
49. swap(A[i], A[maxRow]);
50. swap(b[i], b[maxRow]);
51. // Проверка вырожденности матрицы
52. **if** (fabs(A[i][i]) < 1e-9)
53. {
54. cerr << "Error: Division by zero detected!" << endl;
55. **return**;
56. }
57. // Исключение переменных в нижних строках
58. **for** (int k = i + 1; k < n; k++)
59. {
60. double factor = A[k][i] / A[i][i];
61. **for** (int j = i; j < n; j++)
62. A[k][j] -= factor \* A[i][j];
63. b[k] -= factor \* b[i];
64. }
65. }
66. // Обратная подстановка
67. vector<double> **x**(n);
68. **for** (int i = n - 1; i >= 0; i--)
69. {
70. x[i] = b[i] / A[i][i];
71. **for** (int k = 0; k < i; k++)
72. b[k] -= A[k][i] \* x[i];
73. }
74. cout << "Sequential solution: ";
75. **for** (double xi : x)
76. cout << xi << " ";
77. cout << endl;
78. }
79. /\*\*
80. \* Параллельная реализация метода Гаусса с использованием MPI
81. \* @param n - размер системы
82. \* @param A - матрица коэффициентов
83. \* @param b - вектор правых частей
84. \* @param rank - ранг текущего процесса
85. \* @param size - общее количество процессов
86. \*/
87. void **parallel\_gaussian\_elimination**(int n, vector<vector<double>> &A, vector<double> &b, int rank, int size)
88. {
89. **for** (int i = 0; i < n; i++)
90. {
91. // Главный процесс определяет ведущую строку
92. int maxRow = i;
93. **if** (rank == 0)
94. {
95. **for** (int k = i + 1; k < n; k++)
96. **if** (fabs(A[k][i]) > fabs(A[maxRow][i]))
97. maxRow = k;
98. }
99. // Рассылка номера ведущей строки всем процессам
100. MPI\_Bcast(&maxRow, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);
101. // Синхронизация состояния матрицы
102. **if** (maxRow != i)
103. {
104. swap(A[i], A[maxRow]);
105. swap(b[i], b[maxRow]);
106. }
107. // Проверка на вырожденность системы
108. **if** (fabs(A[i][i]) < 1e-9)
109. {
110. **if** (rank == 0)
111. cerr << "Error: Singular matrix!" << endl;
112. **return**;
113. }
114. // Рассылка ведущей строки всем процессам
115. MPI\_Bcast(A[i].data(), n, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);
116. MPI\_Bcast(&b[i], 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);
117. // Распределение строк между процессами
118. **for** (int j = i + 1; j < n; j++)
119. {
120. **if** (j % size == rank)
121. { // Циклическое распределение строк
122. double factor = A[j][i] / A[i][i];
123. **for** (int k = i; k < n; k++)
124. A[j][k] -= factor \* A[i][k];
125. b[j] -= factor \* b[i];
126. }
127. }
128. }
129. // Обратная подстановка (только на главном процессе)
130. **if** (rank == 0)
131. {
132. vector<double> **x**(n);
133. **for** (int i = n - 1; i >= 0; i--)
134. {
135. x[i] = b[i] / A[i][i];
136. **for** (int j = 0; j < i; j++)
137. b[j] -= A[j][i] \* x[i];
138. }
139. cout << "Parallel solution: ";
140. **for** (double xi : x)
141. cout << xi << " ";
142. cout << endl;
143. }
144. }
145. int **main**(int argc, char \*\*argv)
146. {
147. int rank = 0, size = 1;
148. bool useMPI = false;
149. // Инициализация MPI
150. **if** (MPI\_Init(&argc, &argv) == MPI\_SUCCESS)
151. {
152. useMPI = true;
153. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
154. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);
155. }
156. // Определение размера системы
157. int n = (argc > 1) ? atoi(argv[1]) : (useMPI ? 500 : 3);
158. // Инициализация данных
159. vector<vector<double>> A(n, vector<double>(n));
160. vector<double> **b**(n);
161. // Генерация данных на главном процессе
162. **if** (rank == 0)
163. {
164. srand(time(0));
165. // Создание диагонально доминирующей матрицы
166. **for** (int i = 0; i < n; i++)
167. {
168. double row\_sum = 0;
169. **for** (int j = 0; j < n; j++)
170. {
171. A[i][j] = rand() % 10 + 1;
172. row\_sum += abs(A[i][j]);
173. }
174. A[i][i] = row\_sum; // Гарантия сходимости
175. b[i] = rand() % 20 + 1;
176. }
177. **if** (!useMPI)
178. print\_matrix(A, b);
179. }
180. // Замер времени выполнения
181. double start\_time = useMPI ? MPI\_Wtime() : clock();
182. // Выбор режима вычислений
183. **if** (useMPI)
184. {
185. parallel\_gaussian\_elimination(n, A, b, rank, size);
186. }
187. **else**
188. {
189. sequential\_gaussian\_elimination(A, b);
190. }
191. // Вывод времени выполнения
192. **if** (rank == 0)
193. {
194. double end\_time = useMPI ? MPI\_Wtime() : clock();
195. cout << "Execution time: "
196. << (end\_time - start\_time) / (useMPI ? 1.0 : CLOCKS\_PER\_SEC)
197. << " sec" << endl;
198. }
199. // Завершение работы MPI
200. **if** (useMPI)
201. MPI\_Finalize();
202. **return** 0;
203. }