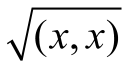
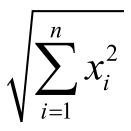
**Задача.**

Разработать программу численного решения СЛАУ методом Гаусса без выбора ведущего элемента и с выбором ведущего элемента по столбцу. Исходная матрица должна содержать случайные числа из диапазона от -1000 до 1000. Правую часть b задается умножением матрицы A (размера n\*n) на вектор x = (m, m+1, ..., n+m–1): b=A\*x, где m - номер в списке группы. Для оценки погрешности вычислений использовалась евклидова (сферическая норма): ||*x*||2==.

Норма вектора невязки 

Относительная погрешность , где – точное решение (*m*, *m*+1, ..., *n*+*m*–1).

**Входные данные.**

n = 1800, m = 4

**Листинг программы.**

1. #include <iostream>
2. #include <vector>
3. #include <stdlib.h>
4. #include <ctime>
5. #include <cmath>
6. #include <chrono>
7. #include <string>
8. int n;
9. std::vector<std::vector<float>> GenerateMatrix(const int size)
10. {
11. std::vector<std::vector<float>> matrix(size, std::vector<float>(size));
12. for (int i = 0; i < size; ++i)
13. {
14. for (int j = 0; j < size; ++j)
15. {
16. matrix[i][j] = -1000.0f + static\_cast<float>(std::rand()) /
17. (static\_cast<float>(RAND\_MAX /
18. (2000.0f)));
19. }
20. }
21. return matrix;
22. }
23. #define M 4
24. std::vector<int> GetVector(const int size)
25. {
26. std::vector<int> vector(size);
27. for (int i = 0; i < size; ++i)
28. {
29. vector[i] = M + i;
30. }
31. return vector;
32. }
33. template <typename T>
34. std::vector<float> MatrixVectorMultiply(const std::vector<std::vector<float>> &matrix, const std::vector<T> &vector)
35. {
36. const int size = vector.size();
37. std::vector<float> result(size);
38. for (int i = 0; i < size; i++)
39. {
40. float sum = 0;
41. for (int j = 0; j < size; j++)
42. {
43. sum += matrix[i][j] \* static\_cast<float>(vector[j]);
44. }
45. result[i] = sum;
46. }
47. return result;
48. }
49. void MakeMove(std::vector<std::vector<float>> &matrix, std::vector<float> &vector, const int k)
50. {
51. for (int i = k + 1; i < n; i++)
52. {
53. float lik = matrix[i][k] / matrix[k][k];
54. matrix[i][k] = 0.0;
55. for (int j = k + 1; j < n; j++)
56. {
57. matrix[i][j] -= lik \* matrix[k][j];
58. }
59. vector[i] -= lik \* vector[k];
60. }
61. }
62. std::vector<float> GetGaussResult(const std::vector<std::vector<float>> &matrix, const std::vector<float> &vector)
63. {
64. std::vector<float> solution(n);
65. solution[n - 1] = vector[n - 1] / matrix[n - 1][n - 1];
66. for (int i = n - 2; i >= 0; --i)
67. {
68. float sum = 0.0;
69. for (int j = i + 1; j < n; ++j)
70. {
71. sum += matrix[i][j] \* solution[j];
72. }
73. solution[i] = (vector[i] - sum) / matrix[i][i];
74. }
75. return solution;
76. }
77. std::vector<float> GaussWithoutSelectingLeadingElement(const std::vector<std::vector<float>> &matrix, const std::vector<float> &vector)
78. {
79. std::vector<std::vector<float>> matrixCopy(matrix);
80. std::vector<float> vectorCopy(vector);
81. for (int k = 0; k < n - 1; ++k)
82. {
83. MakeMove(matrixCopy, vectorCopy, k);
84. }
85. return GetGaussResult(matrixCopy, vectorCopy);
86. }
87. int FindRowWithMaxElement(const std::vector<std::vector<float>> &matrix, const int startRow)
88. {
89. const int k = startRow;
90. int maxRowIndex = startRow;
91. float maxElement = std::abs(matrix[k][k]);
92. for (int i = k + 1; i < matrix.size(); i++)
93. {
94. float absElement = std::abs(matrix[i][k]);
95. if (absElement > maxElement)
96. {
97. maxRowIndex = i;
98. maxElement = absElement;
99. }
100. }
101. return maxRowIndex;
102. }
103. void SwapRows(std::vector<std::vector<float>> &matrix, std::vector<float> &vector, const int i, const int j)
104. {
105. std::swap(matrix[i], matrix[j]);
106. std::swap(vector[i], vector[j]);
107. }
108. std::vector<float> GaussWithSelectingLeadingElement(const std::vector<std::vector<float>> &matrix, const std::vector<float> &vector)
109. {
110. std::vector<std::vector<float>> matrixCopy(matrix);
111. std::vector<float> vectorCopy(vector);
112. for (int k = 0; k < n - 1; k++)
113. {
114. const int max = FindRowWithMaxElement(matrixCopy, k);
115. SwapRows(matrixCopy, vectorCopy, k, max);
116. MakeMove(matrixCopy, vectorCopy, k);
117. }
118. return GetGaussResult(matrixCopy, vectorCopy);
119. }
120. template <typename T>
121. void PrintVector(const std::vector<T> &vector, const int numberOfElements = 0, const std::string &message = "")
122. {
123. if (message != "")
124. {
125. std::cout << message << ' ';
126. }
127. const int border = numberOfElements != 0 ? numberOfElements : vector.size();
128. for (int i = 0; i < border; ++i)
129. {
130. std::cout << vector[i] << ' ';
131. }
132. std::cout << '\n';
133. }
134. template <typename T>
135. float CalculateEuclideanNorm(const std::vector<T> &vector)
136. {
137. float sumOfSquares = 0.0;
138. for (auto element : vector)
139. {
140. sumOfSquares += static\_cast<float>(element \* element);
141. }
142. return std::sqrt(sumOfSquares);
143. }
144. float GetNormOfResidualVector(const std::vector<std::vector<float>> &matrix, const std::vector<float> &calculatedVector, const std::vector<float> &f)
145. {
146. std::vector<float> ax = MatrixVectorMultiply(matrix, calculatedVector);
147. for (int i = 0; i < n; ++i)
148. {
149. ax[i] = f[i] - ax[i];
150. }
151. return CalculateEuclideanNorm(ax);
152. }
153. float GetRelativeError(const std::vector<int> &originalVector, const std::vector<float> &calculatedVector)
154. {
155. std::vector<float> temp(n);
156. for (int i = 0; i < n; ++i)
157. {
158. temp[i] = originalVector[i] - calculatedVector[i];
159. }
160. return CalculateEuclideanNorm(temp) / CalculateEuclideanNorm(originalVector);
161. }
162. int main()
163. {
164. std::ios\_base::sync\_with\_stdio(false);
165. std::cin.tie(nullptr);
166. std::srand(static\_cast<unsigned int>(std::time(nullptr)));
167. std::cin >> n;
168. const std::vector<std::vector<float>> matrix = GenerateMatrix(n);
169. const std::vector<int> vector = GetVector(n);
170. const std::vector<float> b = MatrixVectorMultiply(matrix, vector);
171. auto start = std::chrono::steady\_clock::now();
172. const std::vector<float> res1 = GaussWithoutSelectingLeadingElement(matrix, b);
173. auto end = std::chrono::steady\_clock::now();
174. const int time1 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();
175. start = end;
176. const std::vector<float> res2 = GaussWithSelectingLeadingElement(matrix, b);
177. end = std::chrono::steady\_clock::now();
178. const int time2 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();
179. PrintVector(vector, 5, "Original vector x:");
180. PrintVector(res1, 5, "Vector x calculated without selecting leading element:");
181. PrintVector(res2, 5, "Vector x calculated with selecting leading element:");
182. std::cout << "Norm of residual vector (result calculated without selecting leading element): " << GetNormOfResidualVector(matrix, res1, b) << '\n';
183. std::cout << "Norm of residual vector (result calculated with selecting leading element): " << GetNormOfResidualVector(matrix, res2, b) << '\n';
184. std::cout << "RelativeError (result calculated without selecting leading element): " << GetRelativeError(vector, res1) << '\n';
185. std::cout << "RelativeError (result calculated with selecting leading element): " << GetRelativeError(vector, res2) << '\n';
186. std::cout << "Time (result calculated without selecting leading element): " << time1 << "ms\n";
187. std::cout << "Time (result calculated with selecting leading element): " << time2 << "ms\n";
188. return 0;
189. }

**Выходные данные.**

Original vector x: 4 5 6 7 8

Vector x calculated without selecting leading element: 144.366 -73.6808 -368.511 288.872 146.74

Vector x calculated with selecting leading element: 4.03089 5.71103 6.37613 7.41799 7.56475

Norm of residual vector (result calculated without selecting leading element): 5.05257e+07

Norm of residual vector (result calculated with selecting leading element): 44623.7

RelativeError (result calculated without selecting leading element): 0.414362

RelativeError (result calculated with selecting leading element): 0.000423037

Time (result calculated without selecting leading element): 13824ms

Time (result calculated with selecting leading element): 10714ms

**Выводы.**

Метод Гаусса является неустойчивым по входным данным: в случае относительной малости главного элемента процесс вычислений приводит к сильному накоплению погрешностей. В то же время метод Гаусса с выбором ведущего элемента является устойчивым.