МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа № 3

«Метод прогонки»

**Работу выполнил:**

Шамына Алексей Артемович

Группа: 9

Минск 2024

# Оглавление

[Оглавление 1](#_Toc17691)

[Постановка задачи 2](#_Toc12801)

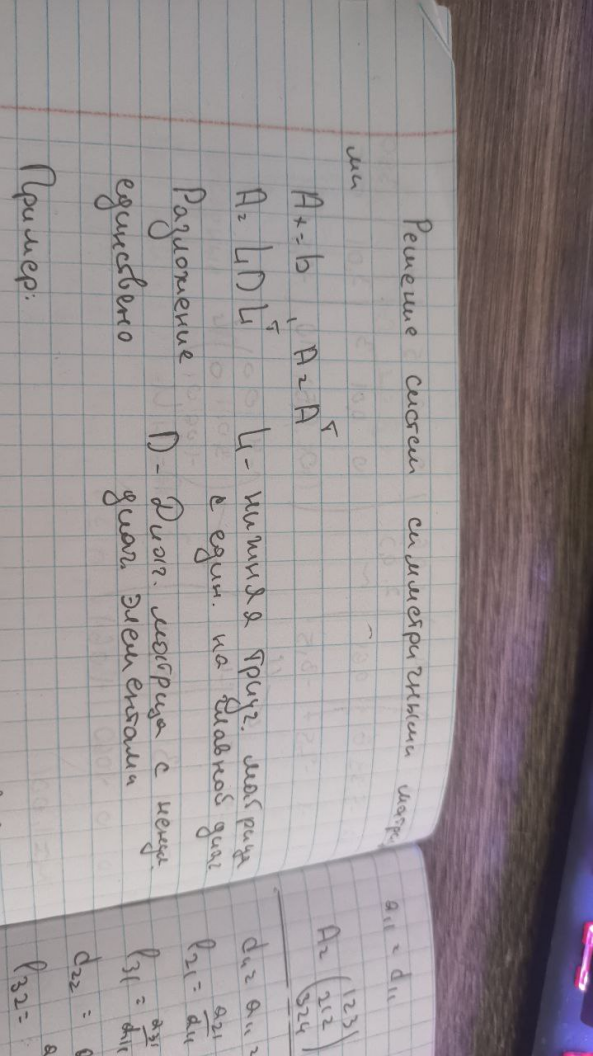
[Краткие теоретические сведенья 3](#_Toc6240)

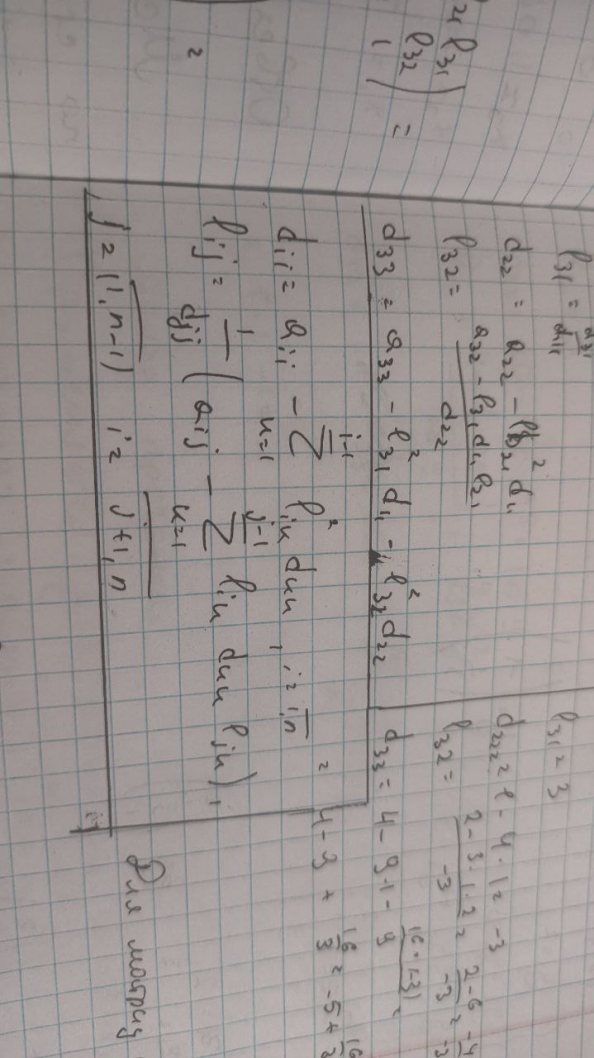
[Листинг 4](#_Toc15102)

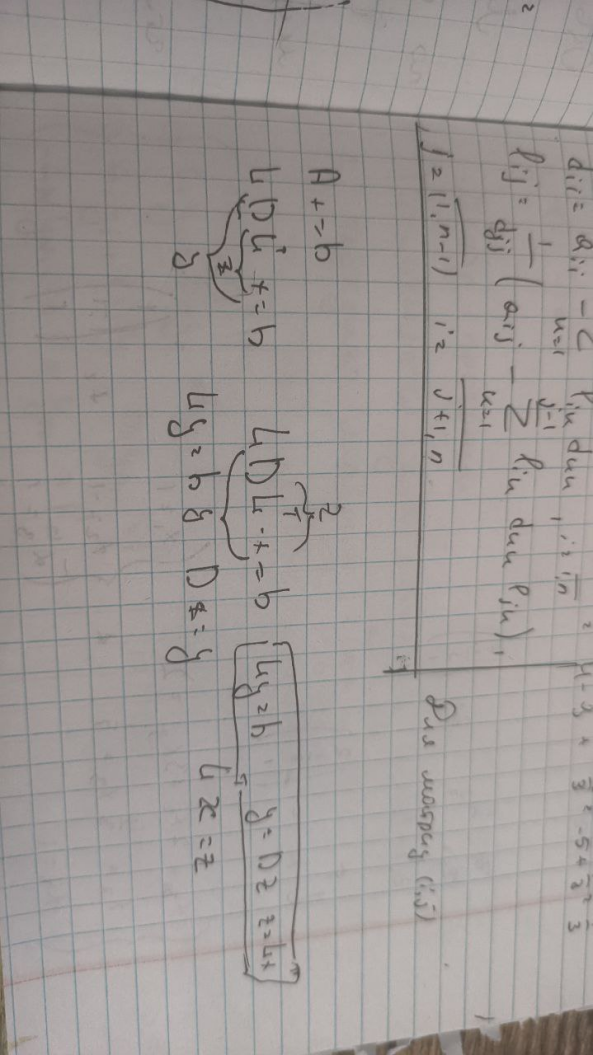
[Результаты 9](#_Toc31529)

# Постановка задачи

# Краткие теоретические сведенья







# Листинг

|  |
| --- |
| for (int i = 0; i < n; i++) // заполняем матрицу А  {  vector <double> temp;  for (int q = 0; q < i; q++)  {  temp.push\_back(rand() % 201 - 100.0);  }  temp.push\_back(0);  matA.push\_back(temp);  }  for (int i = 0; i < n; i++) // заполняем диагонали  {  double sum = 0;  for (int q = 0; q < matA[i].size(); q++)  {  sum += abs(matA[i][q]);  }  for (int q = matA[i].size(); q < matA.size(); q++)  {  sum += abs(matA[q][i]);  }  matA[i][i] = rand() % 120 + 12.0 + sum;  } |

В памяти храним только нижний треугольник матрицы A тк она симметричная

|  |
| --- |
| void mulMat(vector<vector<double>>& matA, vector<double>& x, vector<double>& matB)  {  for (int i = 0; i < matA.size(); i++) // получаем значения b  {  for (int q = 0; q < matA[i].size(); q++)  {  matB[i] += matA[i][q] \* x[q];  }  for (int q = matA[i].size(); q < matA.size(); q++)  {  matB[i] += matA[q][i] \* x[q];  }  }  } |

Умножение матриц

|  |
| --- |
| void LD(vector<vector<double>>& mat)  {  for (int i = 0; i < mat.size(); i++) // идем по столбцам вниз  {  double sum = 0;  for (int k = 0; k < i; k++)  {  sum += mat[i][k] \* mat[i][k] \* mat[k][k];  }  mat[i][i] -= sum;  for (int q = i + 1; q < mat.size(); q++)  {  sum = 0;  for (int k = 0; k < i; k++)  {  sum += mat[q][k] \* mat[i][k] \* mat[k][k];  }  mat[q][i] -= sum;  mat[q][i] /= mat[i][i];  }  }  } |

Разложение матрицы на LD (снова берем только нижнюю треугольную часть тк матрица симметричка)   
Записываем новые значения в ячейки выделенные для матрицы A

|  |
| --- |
| void find\_z(vector<vector<double>>& A, vector<double>& z, vector<double>& b)  {  for (int i = 0; i < A.size(); i++)  {  for (int q = 0; q < i; q++)  {  b[i] -= A[i][q] \* z[q];  }  z[i] = b[i];  }  for (int i = 0; i < A.size(); i++)  {  z[i] /= A[i][i];  }  } |

Находим y и z одной функцией (записываем их значения так же сразу в вектор ответа)

|  |
| --- |
| void find\_x(vector<vector<double>>& A, vector<double>& z)  {  for (int i = A.size() - 1; i >= 0; i--)  {  for (int q = i + 1; q < A.size(); q++)  {  z[i] -= A[q][i] \* z[q];  }  }  } |

Находим значения x (решение СЛАУ)

|  |
| --- |
| void printVec(vector<double>& mat)  {  for (int q = 0; q < mat.size() && q < 5; q++)  {  cout << mat[q] << " ";  }  cout << "\n";  for (int q = mat.size() - 6 > 5 ? mat.size() - 6 : 5;  q < mat.size(); q++)  {  cout << mat[q] << " ";  }  cout << "\n";  } |

Вывод вектора

|  |
| --- |
| auto start = chrono::high\_resolution\_clock::now();  LD(matA); // преобразуем A в LD  find\_z(matA, impreciseSolution, matB); // находим вектор z  find\_x(matA, impreciseSolution); // находим x  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Imprecise solution:\n";  printVec(impreciseSolution);    cout << "\nTime: " << duration.count() << "\n\n"; |

Замер времени и вывод ответов

|  |
| --- |
| double normSol = 0;  for (int i = 0; i < solution.size(); i++)  {  normSol += abs(solution[i]);  }  double normSolImprecise = 0;  for (int i = 0; i < solution.size(); i++)  {  normSolImprecise += abs(solution[i] - impreciseSolution[i]);  }  cout << "Relative error: " << normSolImprecise / normSol << "\n"; |

Вывод относительной погрешности

# Результаты

|  |  |  |
| --- | --- | --- |
| Solution: | Imprecise solution: | Imprecise solution Gauss: |
| 12 | 12.0000000000000071054274 | 11.9999999999999857891453 |
| 13 | 13.0000000000000000000000 | 13.0000000000000000000000 |
| 14 | 13.9999999999999538147222 | 13.9999999999999449329380 |
| 15 | 15.0000000000000000000000 | 15.0000000000000675015599 |
| 16 | 16.0000000000000568434189 | 15.9999999999999893418590 |
| 1007 | 1007.0000000000006821210263 | 1006.9999999999988631316228 |
| 1008 | 1008.0000000000014779288904 | 1007.9999999999993178789737 |
| 1009 | 1009.0000000000003410605132 | 1008.9999999999997726263246 |
| 1010 | 1009.9999999999985220711096 | 1009.9999999999998863131623 |
| 1011 | 1011.0000000000005684341886 | 1010.9999999999997726263246 |
| Time: | 7.4675558999999998022190s | 17.3107576999999999145530s |
| Relative error: | 0.0000000000000012277803 | 0.0000000000000014491981 |

По сравнению с лабораторной 1 время выполнения и относительная погрешность уменьшились

# Вывод

Данный алгоритм работает быстрее и с меньшей погрешностью чем алгоритм Гаусса с выбором главного элемента по столбцу. Я предполагаю, что ускорение алгоритма связано с тем, что мы не тратим время на поиск главного элемента и перестановку строк. Кроме этого алгоритм позволяет экономить память, тк мы можем хранить только нижний треугольник матрицы A.