Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**Решение задач линейной алгебры**

Лабораторная работа №2 по дисциплине

«Вычислительная математика»

Выполнил:

Студент гр. 434-1

Ю.А. Богомолов

Проверил:

Доцент каф. АСУ

Кандидат технических наук

В.В. Романенко

Томск, 2016

Оглавление

[1 Введение 2](#_Toc466903319)

[2 Основная часть 3](#_Toc466903320)

[2.1 Формат входных и выходных данных 3](#_Toc466903321)

[2.2 Вспомогательные функции и классы 3](#_Toc466903322)

[2.3 Главная функция программы 4](#_Toc466903323)

[2.5 Входные и выходные данные 10](#_Toc466903324)

[3 Заключение 16](#_Toc466903325)

# 1 Введение

В ходе данной лабораторной работы необходимо реализовать один из методов решения системы линейных алгебраических уравнений (СЛАУ) и вычисления различных характеристик матриц – определителя и обратной матрицы. По желанию также можно реализовать и дополнительный метод для решения СЛАУ.

СЛАУ выглядит следующим образом:

*,* (1.1)

где – матрица размером , – вектор неизвестных длиной , – вектор свободных коэффициентов длиной . Все вектора являются столбцами.

По условию задания рассматриваются только нормально определённые системы с (т.е. имеющие квадратную матрицу ).

Точность решения СЛАУ можно оценить, вычислив вектор невязки:

*, (1.2)*

где – приближённое решение СЛАУ.

Вариант моего задания – №1 – реализация метода Гаусса-Жордана. Также был выбран дополнительный итерационный метод – Гаусса-Зейделя.

# 2 Основная часть

## 2.1 Формат входных и выходных данных

Формат входного файла:

– порядок матрицы

– коэффициенты матрицы

– вектор свободных коэффициентов

Формат выходного файла:

Шаги преобразования матрицы к верхнему треугольному виду при методе Гаусса-Жордана, также при этом изменяющийся вектор свободных коэффициентов и преобразовываемая к обратной матрице единичная матрица.

Вывод определителя матрицы.

Пошаговое решение СЛАУ, проверка с помощью нахождения вектора невязки.

Шаги преобразования треугольной матрицы к единичной матрице, также при этому преобразовываемая к обратной матрице нижняя треугольная матрица, полученная из единичной.

Проверка того, что мы получили обратную матрицу, с помощью нахождения матрицы невязки.

Решение СЛАУ методом Гаусса-Зейделя без вывода шагов, т.к. тогда вывод будет слишком большим. Вывод количества потребовавшихся шагов.

Проверка решения СЛАУ с помощью нахождения вектора невязки.

## 2.2 Вспомогательные функции и классы

Для выполнения данной лабораторной работы мне было необходимо воспользоваться классом Матрицы , когда-то уже написанным для сдачи лабораторных работ по дисциплине ООП. Также для удобства был написан класс, позволяющий работать с матрицами, хотя бы одна из сторон которых равна - . Для [более-менее] красивого оформления вывода матричных уравнений потребовалось написать класс, выводящий последовательность матриц и символов разделителей «в одну строчку» – . Их код я приводить не буду, так как это не относится непосредственно к решению поставленной задачи.

Также были необходимы две вспомогательные функции: нахождение нормы матрицы и приведение элементов матрицы, меньших эталона, к нулю. Их код представлен в листинге 2.1.

*Таблица 2.1 – Листинг 2.1*

|  |
| --- |
| Листинг 2.1 – Вспомогательные функции |
| const double EPS = 1e-15;  double norm(const RealMatrix& matrix) {  double result = 0.0;  for (auto iter = matrix.colCBegin(); iter != matrix.colCEnd(); ++iter)  for (auto el : iter)  result += el \* el;  return sqrt(result);  }  void cutSmallToZeros(RealMatrix& matrix, double eps = EPS) {  for (auto iter = matrix.colBegin(); iter != matrix.colEnd(); ++iter)  for (double& el : iter)  if (fabs(el) < eps)  el = 0.0;  } |

## 2.3 Главная функция программы

Всё решение поставленной задачи располагается в функции , которая: считывает входные данные, решает СЛАУ двумя вышеупомянутыми методами, находит определитель и обратную матрицу, выводит результат. Вывод в программе является достаточным комментарием к тому, что там происходит, поэтому отдельных комментариев я не пишу.

*Таблица 2.2 – Листинг 2.2*

|  |
| --- |
| Листинг 2.2 – Главная функция программы |
| int main() {  #ifndef ONLINE\_JUDGE  freopen\_in("input.txt");  freopen\_out("output.txt");  #endif  ios\_base::sync\_with\_stdio(false);  cin.tie(NULL);  size\_t matrixSize;  while (cin >> matrixSize) {  RealMatrix matrix(matrixSize, matrixSize);  for (size\_t i = 0; i < matrixSize; ++i)  for (size\_t j = 0; j < matrixSize; ++j)  cin >> matrix[j][i];  RealMatrix coeffsM(1, matrixSize);  RealVectoredMatrix coeffs(&coeffsM);  for (size\_t i = 0; i < matrixSize; ++i)  cin >> coeffs[i];  RealMatrix invertible(matrixSize, matrixSize);  for (size\_t i = 0; i < matrixSize; ++i)  for (size\_t j = 0; j < matrixSize; ++j)  invertible[i][j] = (double)(i == j);  RealMatrix initialMatrix = matrix,  initialCoeffs = coeffsM,  initialInvertible = invertible;  cout << "================================================================================\n";  RealMatrixStatementOutput output(matrixSize);  output << formatWidth(4) << &matrix << "|" << &coeffsM << "|" << &invertible;  cout << "Initial:\n" << output << '\n';  cout << ".................... Gauss-Jordan method ....................\n\n";  cout << "--------------------------------------------------\n";  cout << "Converting matrix to the triangular one:\n\n";  double det = 1.0;  for (size\_t step = 0; step < matrixSize; ++step) {  cout << "Step #" << step + 1 << ":\n";  if (matrix[step][step] <= EPS) {  for (size\_t row = step + 1; row < matrixSize; ++row) {  if (matrix[step][row] > EPS) {  det \*= -1.0;  for (size\_t id = 0; id < matrixSize; ++id) {  swap(matrix[id][step], matrix[id][row]);  swap(invertible[id][step], invertible[id][row]);  }  swap(coeffs[step], coeffs[row]);  break;  }  }  }  double div = matrix[step][step];  det \*= div;  coeffs[step] /= div;  for (size\_t i = 0; i < matrixSize; ++i) {  invertible[i][step] /= div;  matrix[i][step] /= div;  }  for (size\_t row = step + 1; row < matrixSize; ++row) {  double mult = matrix[step][row] / matrix[step][step];  for (size\_t id = 0; id < matrixSize; ++id) {  matrix[id][row] -= matrix[id][step] \* mult;  invertible[id][row] -= invertible[id][step] \* mult;  }  coeffs[row] -= coeffs[step] \* mult;  }  cout << output << '\n';  }  cout << "--------------------------------------------------\n";  cout << "Determinant is " << det << "\n\n";  cout << "--------------------------------------------------\n";  cout << "Solution of SLAE:\n\n";  RealMatrix xM(matrixSize, 1, nan(""));  RealVectoredMatrix x(&xM);  RealMatrixStatementOutput xOutput(1);  xOutput << "(" << &xM << ")";  for (size\_t i = matrixSize - 1; i != -1; --i) {  cout << "Step #" << matrixSize - i << '\n';  double rhs = coeffs[i];  for (size\_t j = i + 1; j < matrixSize; ++j)  rhs -= matrix[j][i] \* x[j];  rhs /= matrix[i][i];  x[i] = rhs;  cutSmallToZeros(xM);  cout << "x =" << xOutput << "\n";  }  cout << "Residual vector:\n";  RealMatrixStatementOutput residualVectorOutput(matrixSize);  RealMatrix xTM = createTransposed(xM);  RealMatrix residualVector = initialMatrix \* xTM - initialCoeffs;  cutSmallToZeros(residualVector);  residualVectorOutput << formatWidth(4)  << &initialMatrix << " \*" << &xTM << " -" << &initialCoeffs  << " =" << &residualVector;  cout << residualVectorOutput << "\n";  cout << "--------------------------------------------------\n";  cout << "Converting matrix to the diagonal one:\n\n";  output.clear();  output << formatWidth(4) << &matrix << "|" << &invertible;  for (size\_t step = matrixSize - 1; step != 0; --step) {  cout << "Step #" << matrixSize - step << ":\n";  for (size\_t row = step - 1; row != -1; --row) {  double mult = matrix[step][row] / matrix[step][step];  for (size\_t id = 0; id < matrixSize; ++id) {  matrix[id][row] -= matrix[id][step] \* mult;  invertible[id][row] -= invertible[id][step] \* mult;  }  }  cout << output << '\n';  }  RealMatrixStatementOutput residualMatrixOutput(matrixSize);  RealMatrix residualMatrix = initialMatrix \* invertible;  cutSmallToZeros(residualMatrix);  residualMatrixOutput << formatWidth(4)  << &initialMatrix << " \*" << &invertible  << " =" << &residualMatrix;  cout << "Checking that we have the invertible matrix:\n"  << residualMatrixOutput << "\n";  residualMatrixOutput.clear();  residualMatrix -= initialInvertible;  cutSmallToZeros(residualMatrix);  residualMatrixOutput << formatWidth(4)  << &initialMatrix << " \*" << &invertible << " -" << &initialInvertible  << " =" << &residualMatrix;  cout << "Residual matrix:\n" << residualMatrixOutput << "\n\n";  cout << ".................... Gauss-Seidel method ....................\n\n";  cout << "--------------------------------------------------\n";  cout << "Solution of SLAE:\n\n";  output.clear();  matrix = initialMatrix;  coeffsM = initialCoeffs;  bool consistent = true;  for (size\_t i = 0; i < matrixSize; ++i) {  if (fabs(matrix[i][i]) > EPS)  continue;  bool swaped = false;  for (size\_t j = 0; j < matrixSize; ++j) {  if (fabs(matrix[i][j] \* matrix[j][i]) < EPS)  continue;  for (size\_t k = 0; k < matrixSize; ++k)  swap(matrix[k][i], matrix[k][j]);  swap(coeffs[i], coeffs[j]);  swaped = true;  }  if (!swaped) {  consistent = false;  break;  }  }  if (!consistent) {  cout << "Matrix is not consistent, cannot solve SLAE\n\n";  continue;  }  xM.resize(1, matrixSize, 0.0);  x = &xM;  vector<RealMatrix> cM(matrixSize, RealMatrix(1, matrixSize, 0.0));  vector<RealVectoredMatrix> c(matrixSize);  RealMatrix dM(1, matrixSize);  RealVectoredMatrix d = &dM;  output << &xM << "=" << "(";  RealMatrix aiiM(1, matrixSize);  RealVectoredMatrix aii = &aiiM;  for (size\_t i = 0; i < matrixSize; ++i) {  aii[i] = matrix[i][i];  c[i] = &cM[i];  d[i] = coeffs[i];  output << &cM[i] << "+";  }  output << &dM << ")" << "/" << &aiiM;  RealMatrix prevXM = xM;  RealVectoredMatrix p = &prevXM;  size\_t step = 1;  for (; ; ++step) {  debug << "Step #" << step << '\n';  for (size\_t i = 0; i < matrixSize; ++i) {  double tmp = d[i];  for (size\_t j = 0; j < i; ++j)  tmp -= matrix[j][i] \* x[j];  for (size\_t j = i + 1; j < matrixSize; ++j)  tmp -= matrix[j][i] \* p[j];  x[i] = tmp / aii[i];  }  debug << output << '\n';  if (norm(xM - prevXM) <= EPS || step > 100)  break;  swap(prevXM, xM);  }  cout << "Total " << step << " steps\n\n";  xTM = createTransposed(xM);  xOutput.clear();  xOutput << "(" << &xTM << ")";  cout << "x = " << xOutput << '\n';  residualVector = initialMatrix \* xM - initialCoeffs;  cutSmallToZeros(residualVector);  residualVectorOutput.clear();  residualVectorOutput << formatWidth(4)  << &initialMatrix << " \*" << &xM << " -" << &initialCoeffs  << " =" << &residualVector;  cout << "Residual vector:\n" << residualVectorOutput << "\n\n";  }  close\_files;  breakpoint\_counters\_clear;  return 0;  } |

## 2.5 Входные и выходные данные

|  |  |
| --- | --- |
| Листинг 2.3 – Входные и выходные данные | |
| input.txt | output.txt |
| 3  1 0 0  0 0 1  0 1 0  2 6 -18 | ================================================================================  Initial:  1 0 0 | 2 | 1 0 0  0 0 1 | 6 | 0 1 0  0 1 0 | -18 | 0 0 1  .................... Gauss-Jordan method ....................  --------------------------------------------------  Converting matrix to the triangular one:  Step #1:  1 0 0 | 2 | 1 0 0  0 0 1 | 6 | 0 1 0  0 1 0 | -18 | 0 0 1  Step #2:  1 0 0 | 2 | 1 0 0  0 1 0 | -18 | 0 0 1  0 0 1 | 6 | 0 1 0  Step #3:  1 0 0 | 2 | 1 0 0  0 1 0 | -18 | 0 0 1  0 0 1 | 6 | 0 1 0  --------------------------------------------------  Determinant is -1  --------------------------------------------------  Solution of SLAE:  Step #1  x = ( nan nan 6 )  Step #2  x = ( nan -18 6 )  Step #3  x = ( 2 -18 6 )  Residual vector:  1 0 0 2 2 0  0 0 1 \* -18 - 6 = 0  0 1 0 6 -18 0  --------------------------------------------------  Converting matrix to the diagonal one:  Step #1:  1 0 0 | 1 0 0  0 1 0 | 0 0 1  0 0 1 | 0 1 0  Step #2:  1 0 0 | 1 0 0  0 1 0 | 0 0 1  0 0 1 | 0 1 0  Checking that we have the invertible matrix:  1 0 0 1 0 0 1 0 0  0 0 1 \* 0 0 1 = 0 1 0  0 1 0 0 1 0 0 0 1  Residual matrix:  1 0 0 1 0 0 1 0 0 0 0 0  0 0 1 \* 0 0 1 - 0 1 0 = 0 0 0  0 1 0 0 1 0 0 0 1 0 0 0  .................... Gauss-Seidel method ....................  --------------------------------------------------  Solution of SLAE:  Total 2 steps  x = ( 2 -18 6 )  Residual vector:  1 0 0 2 2 0  0 0 1 \* -18 - 6 = 0  0 1 0 6 -18 0 |
| 3  3.8 1.7 2  0.9 2 -0.8  -0.9 0.4 1.5  0.4 1.9 7.4 | ================================================================================  Initial:  3.8 1.7 2 | 0.4 | 1 0 0  0.9 2 -0.8 | 1.9 | 0 1 0  -0.9 0.4 1.5 | 7.4 | 0 0 1  .................... Gauss-Jordan method ....................  --------------------------------------------------  Converting matrix to the triangular one:  Step #1:  1 0.447368 0.526316 | 0.105263 | 0.263158 0 0  0 1.59737 -1.27368 | 1.80526 | -0.236842 1 0  0 0.802632 1.97368 | 7.49474 | 0.236842 0 1  Step #2:  1 0.447368 0.526316 | 0.105263 | 0.263158 0 0  0 1 -0.797364 | 1.13015 | -0.14827 0.62603 0  0 0 2.61367 | 6.58764 | 0.355848 -0.502471 1  Step #3:  1 0.447368 0.526316 | 0.105263 | 0.263158 0 0  0 1 -0.797364 | 1.13015 | -0.14827 0.62603 0  0 0 1 | 2.52045 | 0.136149 -0.192247 0.382603  --------------------------------------------------  Determinant is 15.865  --------------------------------------------------  Solution of SLAE:  Step #1  x = ( nan nan 2.52045 )  Step #2  x = ( nan 3.13987 2.52045 )  Step #3  x = ( -2.62597 3.13987 2.52045 )  Residual vector:  3.8 1.7 2 -2.62597 0.4 0  0.9 2 -0.8 \* 3.13987 - 1.9 = 0  -0.9 0.4 1.5 2.52045 7.4 0  --------------------------------------------------  Converting matrix to the diagonal one:  Step #1:  1 0.447368 0 | 0.191501 0.101183 -0.20137  0 1 0 | -0.0397101 0.472739 0.305074  0 0 1 | 0.136149 -0.192247 0.382603  Step #2:  1 0 0 | 0.209266 -0.110306 -0.337851  0 1 0 | -0.0397101 0.472739 0.305074  0 0 1 | 0.136149 -0.192247 0.382603  Checking that we have the invertible matrix:  3.8 1.7 2 0.209266 -0.110306 -0.337851 1 0 0  0.9 2 -0.8 \* -0.0397101 0.472739 0.305074 = 0 1 0  -0.9 0.4 1.5 0.136149 -0.192247 0.382603 0 0 1  Residual matrix:  3.8 1.7 2 0.209266 -0.110306 -0.337851 1 0 0 0 0 0  0.9 2 -0.8 \* -0.0397101 0.472739 0.305074 - 0 1 0 = 0 0 0  -0.9 0.4 1.5 0.136149 -0.192247 0.382603 0 0 1 0 0 0  .................... Gauss-Seidel method ....................  --------------------------------------------------  Solution of SLAE:  Total 35 steps  x = ( -2.62597 3.13987 2.52045 )  Residual vector:  3.8 1.7 2 -2.62597 0.4 1.22125e-15  0.9 2 -0.8 \* 3.13987 - 1.9 = -1.33227e-15  -0.9 0.4 1.5 2.52045 7.4 0 |
| 3  8 4 2  3 5 1  3 -2 10  10 5 4 | ================================================================================  Initial:  8 4 2 | 10 | 1 0 0  3 5 1 | 5 | 0 1 0  3 -2 10 | 4 | 0 0 1  .................... Gauss-Jordan method ....................  --------------------------------------------------  Converting matrix to the triangular one:  Step #1:  1 0.5 0.25 | 1.25 | 0.125 0 0  0 3.5 0.25 | 1.25 | -0.375 1 0  0 -3.5 9.25 | 0.25 | -0.375 0 1  Step #2:  1 0.5 0.25 | 1.25 | 0.125 0 0  0 1 0.0714286 | 0.357143 | -0.107143 0.285714 0  0 0 9.5 | 1.5 | -0.75 1 1  Step #3:  1 0.5 0.25 | 1.25 | 0.125 0 0  0 1 0.0714286 | 0.357143 | -0.107143 0.285714 0  0 0 1 | 0.157895 | -0.0789474 0.105263 0.105263  --------------------------------------------------  Determinant is 266  --------------------------------------------------  Solution of SLAE:  Step #1  x = ( nan nan 0.157895 )  Step #2  x = ( nan 0.345865 0.157895 )  Step #3  x = ( 1.03759 0.345865 0.157895 )  Residual vector:  8 4 2 1.03759 10 0  3 5 1 \* 0.345865 - 5 = 0  3 -2 10 0.157895 4 0  --------------------------------------------------  Converting matrix to the diagonal one:  Step #1:  1 0.5 0 | 0.144737 -0.0263158 -0.0263158  0 1 0 | -0.101504 0.278195 -0.0075188  0 0 1 | -0.0789474 0.105263 0.105263  Step #2:  1 0 0 | 0.195489 -0.165414 -0.0225564  0 1 0 | -0.101504 0.278195 -0.0075188  0 0 1 | -0.0789474 0.105263 0.105263  Checking that we have the invertible matrix:  8 4 2 0.195489 -0.165414 -0.0225564 1 0 0  3 5 1 \* -0.101504 0.278195 -0.0075188 = 0 1 0  3 -2 10 -0.0789474 0.105263 0.105263 0 0 1  Residual matrix:  8 4 2 0.195489 -0.165414 -0.0225564 1 0 0 0 0 0  3 5 1 \* -0.101504 0.278195 -0.0075188 - 0 1 0 = 0 0 0  3 -2 10 -0.0789474 0.105263 0.105263 0 0 1 0 0 0  .................... Gauss-Seidel method ....................  --------------------------------------------------  Solution of SLAE:  Total 26 steps  x = ( 1.03759 0.345865 0.157895 )  Residual vector:  8 4 2 1.03759 10 0  3 5 1 \* 0.345865 - 5 = 0  3 -2 10 0.157895 4 0 |

К сожалению, выходной файл получается слишком большим, поэтому пришлось сократить количество тестов вполовину.

# 3 Заключение

В ходе выполнения данной лабораторной работы я научился реализовывать методы для решения системы алгебраических уравнений (СЛАУ), а также научился некоторым полезным вещам в ходе написания вспомогательных классов.