**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 1**

по курсу «Численные методы»

Студент: Гаврилов М.С.

Группа: 80-306б

Преподаватель: Ревизников Д.Л.

Вариант: 7

Оценка:

Москва, 2022

1. **Постановка задачи**

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.



1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.



1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.



1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.



1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.



1. **Описание программы**

Функции, осуществляющие решение проставленных задач было решено объединить в одну программу – консольный матричный калькулятор.

Для работы с квадратными матрицами был реализован класс SQmatrix. Для работы со СЛАУ – класс EQsys.

Взаимодействие с пользователем программа может осуществлять в двух режимах – решение одиночных задач (в основном использовался для отладки) и режиме консольного ввода (это скорее режим конечного автомата, пожалуй).

После ввода матрицы программа через текстовый интерфейс предоставляет набор возможных действий с ней. Среди них: Переход к системе уравнений (путем добавления столбца свободных членов), расчет определителя и обратной матрицы, отыскание и проверка корректности LU-разложения, Отыскание и проверка корректности QR-разложения, поиск собственных векторов и собственных значений различными методами.

Также оба основных класса программы имеют опцию переключения режима вывода на трассирующий, в котором многие методы выводят отладочную информацию. Переключение режима трассировки возможно через тот же интерфейс.

Наконец, если необходимо, можно поменять вектор свободных членов или матрицу, не перезапуская программу.

После перехода к системе, программа выводит список методов, которыми та может быть решена. Эти методы: LU – разложение, Метод прогонки, Метод простых итераций, Метод Зейделя (Гаусса-Зейделя)

|  |
| --- |
|  |
| Рис. 1. Окно программы после введения матрицы |

|  |
| --- |
|  |
| Рис. 2. Окно программы после перехода к системе СЛАУ. |

Тип матрицы, по сути, используется лишь для реализации метода прогонки, для которого нужно хранить только три центральные диагонали. Матрицы, которые хранят только три диагонали помечаются отдельным типом, чтобы все методы, которые не поддерживают такое представление, сразу отклоняли попытки запуска их для трехдиагональной матрицы.

1. **Примеры работы программы**
2. Решение СЛАУ с помощью LU разложения.

Входные данные:

|  |
| --- |
| 4  1 -5 -7 1  1 -3 -9 -4  -2 4 2 1  -9 9 5 3  1  -75 -41 18 29  1 |

Вывод:

|  |
| --- |
| solved with LU  result:  7.000  -8.000  2.000  4.000  Для продолжения нажмите любую клавишу . . . |

Трассировка:

|  |
| --- |
| LU decomposition:  square matrix| size: 4 type: normal  1.00 0.00 0.00 0.00  4.50 1.00 0.00 0.00  -0.50 0.33 1.00 0.00  -0.50 0.11 1.62 1.00  square matrix| size: 4 type: normal  -2.00 4.00 2.00 1.00  0.00 -9.00 -4.00 -1.50  0.00 0.00 -4.67 2.00  0.00 0.00 0.00 -6.57  LU decomposition multiplied:  square matrix| size: 4 type: normal  -2.00 4.00 2.00 1.00  -9.00 9.00 5.00 3.00  1.00 -5.00 -7.00 1.00  1.00 -3.00 -9.00 -4.00  | autocheck: correct |

1. Решение СЛАУ методом прогонки

Входные данные:

|  |
| --- |
| -1  5  15 8  2 -15 4  4 11 5  -3 16 -7  3 8  1  92 -84 -77 15 -11  2 |

Вывод:

|  |
| --- |
| solved with TMA  result:  4.000  4.000  -8.000  -1.000  -1.000  Для продолжения нажмите любую клавишу . . . |

1. Решение СЛАУ методом простых итераций и Зейделя

Входные данные:

|  |
| --- |
| 4  29 8 9 -9  -7 -25 0 9  1 6 16 -2  -7 4 -2 17  1  197 -226 -95 -58  3 |

Вывод (метод простых итераций):

|  |
| --- |
| solved with FPI  result:  7.003  6.003  -8.998  -3.002  error:  0.077365  Для продолжения нажмите любую клавишу . . . |

Трассировка (метод простых итераций):

|  |
| --- |
| square matrix| size: 4 type: normal  0.00 -0.28 -0.31 0.31  -0.28 0.00 0.00 0.36  -0.06 -0.38 0.00 0.13  0.41 -0.24 0.12 0.00  steps estimated: 145  step 1 | X: 6.793103 9.040000 -5.937500 -3.411765  step 2 | X: 5.083159 5.909696 -10.178540 -3.440193  step 3 | X: 7.254054 6.378246 -8.901357 -3.906691  step 4 | X: 6.583656 5.602456 -9.271057 -2.972784  step 5 | X: 7.202235 6.126374 -8.821498 -3.109785  step 6 | X: 6.875670 5.903852 -9.073753 -2.925462  step 7 | X: 7.072545 6.061646 -8.946856 -3.037249  step 8 | X: 6.954941 5.966278 -9.032308 -2.978381  step 9 | X: 7.026038 6.020399 -8.981836 -3.014420  step 10 | X: 6.984260 5.987518 -9.011080 -2.991941  step 11 | X: 7.009383 6.007308 -8.993328 -3.004848  step 12 | X: 6.994409 5.995628 -9.003933 -2.997071  step 13 | X: 7.003336 6.002620 -8.997645 -3.001736 | final  norm of alpha: 0.896552  0.008927 0.006992 0.006288 -0.004665  norm of vector: 0.008927 |

Вывод (метод Зейделя):

|  |
| --- |
| solved with GS  result:  6.999  6.000  -9.000  -3.000  error:  0.018060  Для продолжения нажмите любую клавишу . . . |

Трассировка (метод Зейделя):

|  |
| --- |
| steps estimated: 145  step 1 | X: 6.793103 9.040000 -5.937500 -3.411765  step 2 | X: 5.083159 6.388480 -9.077348 -3.889794  step 3 | X: 6.640695 5.780280 -9.006373 -3.097000  step 4 | X: 7.032487 5.955984 -8.997649 -2.975990  step 5 | X: 7.018864 6.003362 -8.999438 -2.992957  step 6 | X: 7.001084 6.002232 -9.000024 -3.000082  step 7 | X: 6.999367 6.000148 -9.000026 -3.000299 | final |

1. Отыскание собственных значений и векторов методом вращений.

Входные данные:

|  |
| --- |
| 3  -6 6 -8  6 -4 9  -8 9 -2  7 |

Вывод:

|  |
| --- |
| eigenvalues:  0.770697 -19.344142 6.573446  eigenvectors:  | 0.762047 0.622570 -0.178021  | -0.595912 0.566727 -0.568955  | -0.253325 0.539655 0.802869 |

Проверка:

|  |
| --- |
| 0.587307 0.479813 -0.137200  0.587307 0.479813 -0.137200  11.527411 -10.962845 11.005938  11.527471 -10.962972 11.005749  -1.665218 3.547390 5.277619  -1.665078 3.547256 5.277753  eigen correct |

1. Отыскание собственных значений методом QR-разложения.

Входные данные:

|  |
| --- |
| 3  9 0 2  -6 4 4  -2 -7 5  8 |

Вывод:

|  |
| --- |
| (10.0295,0) (3.98527,6.08723) (3.98527,-6.08723) |

Трассировка:

|  |
| --- |
| square matrix| size: 3 type: normal  9.00 0.00 2.00  -6.00 4.00 4.00  -2.00 -7.00 5.00  square matrix| size: 3 type: normal  9.41 0.67 5.44  -3.30 3.99 6.87  -0.64 -4.09 4.60  …  …  …  square matrix| size: 3 type: normal  10.05 2.77 0.74  -0.07 5.57 7.61  -0.02 -5.21 2.38  square matrix| size: 3 type: normal  10.03 1.57 -2.43  -0.06 2.90 -8.10  0.01 4.72 5.08  cc (3.98527,6.08723) (3.98527,-6.08723)  eigenvalues:  (10.0295,0) (3.98527,6.08723) (3.98527,-6.08723) |

Более сложная матрица:

|  |
| --- |
| 1  5  1 -5 -7 1 1  1 -3 -9 -4 2  -2 4 2 1 8  -9 9 5 3 11  6 -2 1 0 4 |

Вывод:

|  |
| --- |
| (2.44748,9.24665) (2.44748,-9.24665) (7.63599,0) (-5.10036,0) (-0.430594,0) |

1. **Часть текста программы**

Основные методы, выполняющие поставленные задачи:

|  |
| --- |
| LU – разложение |
| std::pair<std::pair<SQmatrix, SQmatrix>, std::vector<int>> SQmatrix::LUdecomp(std::vector<double>& ft) {  if (type == "tridiagonalCompact") { //может потом сделаю и для этого типа, но сейчас не нужно  throw ERR\_MATR\_TYPE\_MISSMATCH;  }  SQmatrix L(size);  SQmatrix U((\*this));  for (int i = 0; i < size; ++i) {  //элементы на диагонали = 1  L[i][i] = 1;  }  for (int j = 0; j < size - 1; ++j) {  //Выполняется приведение матрицы U к верх.треуг. виду.  //Коэфициенты, на которые для этого умножаются строки записываются в L.  int k = 0;  //находим строку, в которой на том месте, которое нам нужно наибольший элемент  double maxval = abs(U[k][j]);  for (int i = 1; i< size; ++i) {  if (abs(U[j][i]) > maxval) {  maxval = abs(U[j][i]);  k = j;  }  }  if (maxval < EPS) {  throw ERR\_MATR\_JKB\_NOT\_CASTABLE;  }  U.swap(k, i);  VecSwap(arrmap, k, i);  std::swap(ft[k], ft[i]);  for (int i = j + 1; i < size; ++i) {  //вычисление коэфициента, на который надо умножить строку U[j],  //чтобы при сложении с U[i] эл-т U[i][j] обратился в 0  double fst = U[i][j] / U[j][j];  //запись коэфициента в L  L[i][j] = fst;  //выполнение сложения  U[i] = U[i] - (U[j] \* fst);  }  }  return std::pair<std::pair<SQmatrix, SQmatrix>,std::vector<int>>(std::pair<SQmatrix, SQmatrix>(L, U), arrmap);  } |

|  |
| --- |
| Решение системы с помощью LU-разложения |
| std::vector<double> EQsys::LUsolve() {  if (matr.typef() == "tridiagonalCompact") {  throw EQSYS\_SOLVE\_WRONG\_TYPE;  }  std::vector<double> RSMfreeTerms = freeTerms;  std::pair<std::pair<SQmatrix, SQmatrix>,std::vector<int>> preres = matr.LUdecomp(RSMfreeTerms);  std::pair<SQmatrix, SQmatrix> LUres = preres.first;  std::vector<int> resMap = preres.second;  if (trace) {  printf("LU decomp. results:\n");  LUres.first.print();  LUres.second.print();  }  //проверка правильности LU разложения  if (!matr.LUcheck(LUres, 0.01, resMap)) {  throw EQSYS\_LUSOL\_LUERROR;  }  std::vector<double> tmp(varCnt, 0);  std::vector<double> res(varCnt, 0);  for (int i = 0; i < varCnt; ++i) {  double sumTmp = 0;  for (int j = 0; j < i; ++j) {  sumTmp += LUres.first[i][j] \* tmp[j];  }  tmp[i] = RSMfreeTerms[i] - sumTmp;  }  for (int i = varCnt - 1; i >= 0; --i) {  double sumTmp = 0;  for (int j = i; j < varCnt; ++j) {  sumTmp += LUres.second[i][j] \* res[j];  }  res[i] = (tmp[i] - sumTmp) / (LUres.second[i][i]);  }  std::vector<double> realRes(varCnt);  for (int i = varCnt - 1; i >= 0; --i) {  //переставляем результат в правильном порядке  realRes[resMap[i]] = res[i];  }  return realRes;  } |

|  |
| --- |
| Решение системы с трехдиагональной матрицей методом прогонки |
| std::vector<double> EQsys::TMAsolve() {  //метод прогонки для компактной записи (3 вектора с коэфициентами на диагоналях)  if (matr.typef() != "tridiagonalCompact") {  throw EQSYS\_SOLVE\_WRONG\_TYPE;  }  //первый вектор в пачке -- нижний, "a", второй -- центральный, "b", третий -- верхний, "c"  std::vector<double> alpha(varCnt);  std::vector<double> betha(varCnt);  //отыскание коэфициентов  alpha[0] = - matr[2][0] / matr[1][0];  betha[0] = freeTerms[0] / matr[1][0];  for (int i = 1; i < varCnt - 1; ++i) {  alpha[i] = - matr[2][i] / (matr[1][i] + matr[0][i - 1] \* alpha[i - 1]);  betha[i] = (freeTerms[i] - matr[0][i - 1] \* betha[i - 1]) / (matr[1][i] + matr[0][i - 1] \* alpha[i - 1]);  //первый вектор берется по коэфициентам i-1, т.к. из-за структуры матрицы он записан с увеличением коэфициента на 1  }  //пояснение к коэфициентам векторов  /\*  1 1 0 0 0 0 ...  1 2 2 0 0 0 ...  0 2 3 3 0 0 ...  ........... ...  \*/  std::vector<double> res(varCnt, 0);  //значение последнего неизвестного  res[varCnt - 1] = (freeTerms[varCnt - 1] - matr[0][varCnt - 2] \* betha[varCnt - 2]) / (matr[1][varCnt - 1] + matr[0][varCnt - 2] \* alpha[varCnt - 2]);  //отычкание остальных неизвестных  for (int i = varCnt - 2; i >= 0; --i) {  res[i] = res[i + 1] \* alpha[i] + betha[i];  }  return res;  } |

|  |
| --- |
| Решение системы методом свободных итераций |
| std::pair<std::vector<double>, double> EQsys::FPIsolve(double eps) {  if (matr.typef() == "tridiagonalCompact") {  throw EQSYS\_SOLVE\_WRONG\_TYPE;  }  //алгоритм простых итераций (fixed point iteration)  if (!matr.JkbCompatable()) { //проверка сходимости  throw EQSYS\_SOLVE\_FPI\_NOT\_APPLYABLE;  }  std::pair<SQmatrix, std::vector<double>> casted = matr.JkbCast(freeTerms); //выполнили преобразование  std::vector<double> res = casted.second;  std::vector<double> rp = res - std::vector<double>(res.size(), 1); //значния res на пердыдущем шаге.  if (trace) {  casted.first.print();  }  double alphaNorm = casted.first.norm();//лучше максимальная сумма строки  double stepsEst = (log(eps) - VNorm(casted.second) + log(1 - alphaNorm)) / log(alphaNorm) - 1;  if(trace) printf("steps estimated: %.0lf\n", stepsEst);  int cnt = 0;  while (VNorm(res - rp) > eps) {  ++cnt;  if (trace) {  printf("step %d | X: ", cnt);  for (int i = 0; i < varCnt; ++i) {  printf("%lf ", res[i]);  }  printf("\n");  }  rp = res;  res = casted.second + casted.first.mult(casted.first, res);  }  ++cnt;  if (trace) {  printf("step %d | X: ", cnt);  for (int i = 0; i < varCnt; ++i) {  printf("%lf ", res[i]);  }  printf("| final\n");  printf("norm of alpha: %lf\n", alphaNorm);  VecPrint(res - rp);  printf("norm of vector: %lf\n", VNorm(res - rp));  }  double error = VNorm(res - rp) \* alphaNorm / (1 - alphaNorm);  return std::pair<std::vector<double>, double> (res,error);  } |

|  |
| --- |
| Решение системы методом Гаусса-Зейделя |
| //метод Гаусса-Зейделя  std::pair<std::vector<double>, double> EQsys::GSsolve(double eps) {  if (matr.typef() == "tridiagonalCompact") {  throw EQSYS\_SOLVE\_WRONG\_TYPE;  }  if (!matr.JkbCompatable()) { //проверка сходимости  throw EQSYS\_SOLVE\_FPI\_NOT\_APPLYABLE;  }  std::pair<SQmatrix, std::vector<double>> casted = matr.JkbCast(freeTerms); //выполнили преобразование  std::pair<SQmatrix, SQmatrix> sliced = casted.first.slice();  SQmatrix A = sliced.first.dif(SQmatrix(varCnt, "ident"), sliced.first).inverse();  SQmatrix B = sliced.second;  std::vector<double> res = casted.second;  std::vector<double> rp = res - std::vector<double>(res.size(), 1); //значния res на пердыдущем шаге.  SQmatrix Alpha = A.mult(A, B);  std::vector<double> Betha = A.mult(A, casted.second);  double alphaNorm = Alpha.norm();  double stepsEst = (log(eps) - VNorm(Betha) + log(1 - alphaNorm)) / log(alphaNorm) - 1;  if (trace) printf("steps estimated: %.0lf\n", stepsEst);  int cnt = 0;  while (VNorm(res - rp) > eps) {  ++cnt;  if (trace) {  printf("step %d | X: ", cnt);  for (int i = 0; i < varCnt; ++i) {  printf("%lf ", res[i]);  }  printf("\n");  }  rp = res;  res = Betha + casted.first.mult(Alpha, res);  }  ++cnt;  if (trace) {  printf("step %d | X: ", cnt);  for (int i = 0; i < varCnt; ++i) {  printf("%lf ", res[i]);  }  printf("| final\n");  }  double error = VNorm(res - rp) \* B.norm() / (1 - Alpha.norm());  return std::pair<std::vector<double>, double> (res,error);  } |

|  |
| --- |
| Отыскание собственных векторов и собственных значений матрицы методом вращений |
| //метод вращений Якоби (отыскание собственных значений и собственных векторов методом вращений  std::pair <std::vector<double>, std::vector<std::vector<double>>> SQmatrix::JRM(double acc) {  if (type == "tridiagonalCompact") { //может потом сделаю и для этого типа, но сейчас не нужно  throw ERR\_MATR\_TYPE\_MISSMATCH;  }  //добавить проверку на симметричность  std::pair<int, int>maxPos = maxND();  double angle = (0.5) \* atan(2 \* matrix[maxPos.first][maxPos.second]/(matrix[maxPos.first][maxPos.first] - matrix[maxPos.second][maxPos.second]));  SQmatrix rot = rotation(maxPos.first, maxPos.second, angle);  //применение ротации к матрице  SQmatrix cur = mult(rot.transpose(), \*this);  cur = mult(cur, rot);  if (trace) {  printf("max elm pos:\n");  printf("%d %d\n", maxPos.first, maxPos.second);  rot.print();  cur.print();  }  SQmatrix eigen = rot;  int i = 0;  while (cur.nonDiagSqSm() > acc) {  ++i;  std::pair<int, int>maxPos = cur.maxND();  if (trace) {  printf("max elm pos:\n");  printf("%d %d\n", maxPos.first, maxPos.second);  }  angle = (0.5) \* atan(2 \* cur[maxPos.first][maxPos.second] / (cur[maxPos.first][maxPos.first] - cur[maxPos.second][maxPos.second]));  rot = rotation(maxPos.first, maxPos.second, angle);  //применение ротации к матрице  cur = mult(rot.transpose(), cur);  cur = mult(cur, rot);  eigen = mult(eigen, rot);  if (trace) {  rot.print();  cur.print();  }  if (i > LIMIT) {  throw ERR\_MATR\_JRM\_NO\_CONVERGE;  }  }  SQmatrix eigenT = eigen.transpose();  return std::pair<std::vector<double>, std::vector<std::vector<double>>>(cur.diagonal(), eigenT.matrixf());  } |

|  |
| --- |
| Отыскание собственных значений через QR-разложение |
| //отыскание собственных значений с помощью QR-разложения  std::vector<std::complex<double>> SQmatrix::QReigen(double prec){  if (type == "tridiagonalCompact") { //может потом сделаю и для этого типа, но сейчас не нужно  throw ERR\_MATR\_TYPE\_MISSMATCH;  }  std::pair<SQmatrix, SQmatrix> QR = QRdecomp();  SQmatrix Q = QR.first,  R = QR.second;  SQmatrix RmQ = mult(R, Q);  int i = 0;  while (RmQ.underDiagSqSm() > prec) {  ++i;  QR = RmQ.QRdecomp();  RmQ = mult(QR.second, QR.first);  if(trace) RmQ.print();  if (i > LIMIT) {  throw ERR\_MATR\_QR\_NO\_CONVERGE;  }  }  std::vector<std::complex<double>> eigenvalues(size);  for (int i = 0; i < size; ++i) {  if (i == size - 1) {  eigenvalues[i] = RmQ[i][i];  break;  }  if (pow(RmQ[i + 1][i],2) < prec) {  eigenvalues[i] = RmQ[i][i];  }  else {//если найден блок, то надо вычислить два сз, соответствующих этому блоку и пропустить следующую ячейку  std::pair<std::complex<double>, std::complex<double>> val = solveSQ(1, -(RmQ[i + 1][i + 1] + RmQ[i][i]),  RmQ[i + 1][i + 1] \* RmQ[i][i] - RmQ[i + 1][i] \* RmQ[i][i + 1]);  if (trace) std::cout << " cc " << val.first << " " << val.second << "\n";  eigenvalues[i] = val.first;  eigenvalues[i + 1] = val.second;  ++i;  }  }  return eigenvalues;  } |

1. **Вывод**

В ходе выполнения этой лабораторной работы я получил опыт разработки комплексных, обширных программ на c++, а также в реализации интерфейса взаимодействия с пользователем. Помимо этого, я реализовал большое количество методов обработки матриц и векторов, а также, решения СЛАУ.