Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Численные методы

Практическая работа №3

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студент: Кожекин М.В.

Преподаватели: Задорожный А. Г.

Персова М.Г.

Новосибирск

2018

1. Цель работы

Изучить особенности реализации трёхшаговых итерационных методов для СЛАУ с разреженными матрицами. Исследовать влияние предобусловливания на сходимость изучаемых методов на нескольких матрицах большой (не менее 10000) размерности.

**Вариант 6**: Локально-оптимальная схема для несимметричной матрицы. Факторизация LU(sq).

1. Анализ

Выбирается начальное приближение  и полагается







Далее для  производятся следующие вычисления:



    

где  – вспомогательный вектор, который вычисляется не умножением матрицы на вектор, а пересчитывается рекуррентно.

Можно показать, что при использовании матриц неполной факторизации  и  (соответственно нижняя и верхняя треугольные матрицы неполной факторизации исходной матрицы ), локально оптимальная схема (3.25)–(3.32) (при применении ее к СЛАУ с матрицей ) преобразуется к следующему виду.

Выбирается начальное приближение  и полагается







Далее для  производятся следующие вычисления:













Выход из процесса можно организовать по норме относительной невязки и по шагу.

Разложение LUsq = A имеет следующий вид:

Матрица А должна положительно определённой, что следует из формул разложения:

**Остальные формулы:**

* Прямой обход
* Обратный обход

1. Исследования на матрице с диагональным преобладанием

**Матрица A:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | -1 | 0 | 0 | 0 | -2 | -3 | 0 | 0 | 0 |  | 1 |  | -28 |
| -2 | 9 | -4 | 0 | 0 | 0 | -1 | -2 | 0 | 0 |  | 2 |  | -19 |
| 0 | 0 | 7 | -4 | 0 | 0 | 0 | -2 | -1 | 0 |  | 3 |  | -20 |
| 0 | 0 | -4 | 13 | -3 | 0 | 0 | 0 | -2 | -4 |  | 4 |  | -33 |
| 0 | 0 | 0 | -3 | 6 | -2 | 0 | 0 | 0 | -1 | \* | 5 | = | -4 |
| -1 | 0 | 0 | 0 | -4 | 5 | 0 | 0 | 0 | 0 | 6 | 9 |
| 0 | -4 | 0 | 0 | 0 | 0 | 5 | -1 | 0 | 0 |  | 7 |  | 19 |
| 0 | -1 | -3 | 0 | 0 | 0 | -2 | 7 | -1 | 0 |  | 8 |  | 22 |
| 0 | 0 | -3 | -2 | 0 | 0 | 0 | -1 | 6 | 0 |  | 9 |  | 29 |
| 0 | 0 | 0 | -2 | -1 | 0 | 0 | 0 | -2 | 5 |  | 10 |  | 19 |

**Сравнение методов:**

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 36 | 87 | 1.1951631020067435e-01 |
| LOS + diag | 31 | 121 | 1.3510187796629339e-02 |
| LOS + LU(sq) | 27 | 110 | 1.3896886947002099e-01 |
| Гаусс-Зейдель | 368 | 52 | 8.92218e-11 |

**Результат:**

|  |  |  |
| --- | --- | --- |
| ЛОС | ЛОС + diag | ЛОС + LU(sq) |
| -3.8479794551408943  -3.6007303077764932  -2.8601421076610580  -1.8732920149572483  -0.8326614909701502  0.3363292182248062  1.3492249947438317  2.2257657801087878  3.1324516992180453  4.1093354659720136 | -3.4610008991866175  -3.1623063694195062  -2.4031715320642113  -1.4167916593613392  -0.3653950042013192  0.8012136625279839  1.7948063309146856  2.6787677927053677  3.5930814195271421  4.5773840479308321 | -3.6150097078215948  -3.3366238530177528  -2.5682759571407257  -1.5944920719647848  -0.5468770754383668  0.6012973686004954  1.5914305679977652  2.4803746307553549  3.4002377141688473  4.3810035925507842 |

**Вывод**: происходит выход по шагу, т.к. метод не устойчив на данной матрице

**Матрица B:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 1 | 0 | 0 | 0 | 2 | 3 | 0 | 0 | 0 |  | 1 |  | 42 |
| 2 | 9 | 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 |  | 2 |  | 55 |
| 0 | 0 | 7 | 4 | 0 | 0 | 0 | 2 | 1 | 0 |  | 3 |  | 62 |
| 0 | 0 | 4 | 13 | 3 | 0 | 0 | 0 | 2 | 4 |  | 4 |  | 137 |
| 0 | 0 | 0 | 3 | 6 | 2 | 0 | 0 | 0 | 1 | \* | 5 | = | 64 |
| 1 | 0 | 0 | 0 | 4 | 5 | 0 | 0 | 0 | 0 | 6 | 51 |
| 0 | 4 | 0 | 0 | 0 | 0 | 5 | 1 | 0 | 0 |  | 7 |  | 51 |
| 0 | 1 | 3 | 0 | 0 | 0 | 2 | 7 | 1 | 0 |  | 8 |  | 90 |
| 0 | 0 | 3 | 2 | 0 | 0 | 0 | 1 | 6 | 0 |  | 9 |  | 79 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 2 | 5 |  | 10 |  | 81 |

**Сравнение методов:**

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 43 | 51 | 2.9717076472420172e-16 |
| LOS + diag | 34 | 60 | 5.4612844374616470e-16 |
| LOS + LU(sq) | 9 | 27 | 8.0883721911317692e-16 |
| Гаусс-Зейдель | 21 | 3 | 8.56651e-11 |

**Результат:**

|  |  |  |
| --- | --- | --- |
| ЛОС | ЛОС + diag | ЛОС + LU(sq) |
| 0.9999999997257614  2.0000000000685660  2.9999999973882785  4.0000000014854136  4.9999999971429965  6.0000000027227447  7.0000000002622711  8.0000000022680293  8.9999999993412008  10.0000000006557013 | 0.9999999985587995  2.0000000001424310  2.9999999879836565  4.0000000072016979  4.9999999896441478  6.0000000093366541  6.9999999998263442  8.0000000089289198  9.0000000009928502  9.9999999994548698 | 0.9999999981868053  1.9999999965236945  3.0000000048610276  3.9999999954997643  5.0000000017843949  5.9999999985623047  7.0000000052129767  7.9999999891807176  9.0000000005857039  9.9999999982579038 |

1. Исследование на матрице Гильберта

Размерность матрицы: 4

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 7 | 11 | 8.5560482027294059e-41 |
| LOS + diag | 7 | 12 | 3.1190059370947037e-43 |
| LOS + LU(sq) | 2 | 13 | 1.4861646610310086e-61 |

Размерность матрицы: 7

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 18 | 24 | 6.2912460078970593e-39 |
| LOS + diag | 16 | 25 | 2.1809465921464475e-38 |
| LOS + LU(sq) | 3 | 12 | 9.4053950450887990e-62 |

Размерность матрицы: 10

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 60 | 74 | 3.7951471106617626e-41 |
| LOS + diag | 60 | 194 | 2.5761723928040929e-39 |
| LOS + LU(sq) | 3 | 16 | 1.7973819358951991e-54 |

1. Тесты с большой размерностью

**0945**

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 416 | 20222 | 7.0153309415727521e-21 |
| LOS + diag | 44 | 2589 | 8.7758789218854143e-23 |
| LOS + LU(sq) | 8 | 1327 | 2.3653994181649045e-25 |

**4545**

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Количество итераций | Время решения, мкс | Невязка |
| LOS | 1982 | 501143 | 3.7057100205773134e-20 |
| LOS + diag | 156 | 46491 | 9.7364037730067872e-23 |
| LOS + LU(sq) | 8 | 6851 | 3.0559498009255747e-24 |

1. Вывод:

Как видно из тестов решение СЛАУ при помощи локально-оптимальной схемы с помощью неполной факторизации LU(sq) быстрее и точнее, чем с помощью диагональной факторизации или решение без предобуславливания.

1. Текст программы

Для удобства программа была разбита на следующие модули:

head.h – заголовочный файл, в котором определяется точность вычислений

slae.h и slae.cpp – класс СЛАУ

main.cpp – файл с исследованиями

**head.h**

|  |
| --- |
| #pragma once  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <fstream>  #include <iostream>  #include <vector>  #include <string>  #include <iomanip>  #include <chrono>  using namespace std;  // float || double  typedef double real;  typedef std::vector<real> vec;  // Умножение на константу  inline bool operator==(const vec& a, const vec& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  for (int i = 0; i < a.size(); ++i)  if (a[i] != b[i])  return false;  return true;  }  // Сложение векторов  inline vec operator+(const vec& a, const vec& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  vec result = a;  for (int i = 0; i < b.size(); i++)  result[i] += b[i];  return result;  }  // Вычитание векторов  inline vec operator-(const vec& a, const vec& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  vec result = a;  for (int i = 0; i < b.size(); i++)  result[i] -= b[i];  return result;  }  // Умножение на константу  inline vec operator\*(const vec& a, double b) {  vec result = a;  for (int i = 0; i < result.size(); i++)  result[i] \*= b;  return result;  }  // Умножение на константу  inline vec operator\*(double b, const vec& a) {  return operator\*(a, b);  }  // Скалярное произведение  inline real operator\*(const vec& a, const vec& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  real sum = 0;  for (int i = 0; i < a.size(); i++)  sum += a[i] \* b[i];  return sum;  }  // Потоковый вывод  inline std::ostream& operator<<(std::ostream& out, const vec& v) {  for (int i = 0; i < v.size() - 1; ++i)  out << v[i] << "\t";  out << v.back();  return out;  } |

**SLAE.h**

|  |
| --- |
| #include "head.h"  // Система линейных алгебраических уравнений  class SLAE {  public:  int readSLAEfromFiles(const string &folderName, bool firstNumberIsOne);  void writeSLAEToFiles(const string &folderName);  void writeDenseMatrixLToFile(std::ofstream& fout, const char \*str);  void writeDenseMatrixUToFile(std::ofstream& fout, const char \*str);  void convAToDense();  void convLUToDense();  void multAAndX();  void getVectX(vec &x) { x = F; };  void generateVectX(int size);  void writeXToFile(const char \*fileName);  void writeXToStream(std::ofstream& fout);  void writeFToFile(const char \*fileName);  int getDimention() { return n; }  void decomposionD();  void decomposionChol();  void decomposionLUsq();  vec execDirectTraversal(const vec &\_F);  vec execReverseTraversal(const vec &\_y);  void createHilbertMatrix(int size);  void createHilbertMatricies(int a, int b, int step, const string &folderNameTemplate);  pair<int, real> LOS();  pair<int, real> LOSfactD();  pair<int, real> LOSfactLUsq();  void clearAll();  void setMaxiter(int new\_maxiter) { maxiter = new\_maxiter; }  void setE(real new\_E) { E = new\_E; }  private:  vec multA(const vec&x);  vec multD(const vec&x);  real calcRelativeDiscrepancy();  real calcNormE(vec &x);  vector <vector <double>> L, U;  vec di, al, au, di\_f, al\_f, au\_f;  vector <int> ia, ja;  int n, maxiter;  real E;  vec x, r, z, p, F, Ftmp;  }; |

**SLAE.cpp**

|  |
| --- |
| #include "head.h"  #include "slae.h"  // Ввод СЛАУ: матрицы A и вектора y  int SLAE::readSLAEfromFiles(const string &folderName, bool firstNumberIsOne) {  std::ifstream fin;  fin.open(folderName + "/" + "kuslau.txt");  fin >> n >> maxiter >> E;  fin.close();  x.resize(n, 0);  r.resize(n, 0);  z.resize(n, 0);  p.resize(n, 0);  F.resize(n, 0);  Ftmp.resize(n, 0);  fin.open(folderName + "/" + "di.txt");  di.resize(n);  for (int i = 0; i < di.size(); ++i) {  fin >> di[i];  }  fin.close();  fin.open(folderName + "/" + "ig.txt");  ia.resize(n + 1);  for (int i = 0; i < ia.size(); ++i) {  fin >> ia[i];  if (firstNumberIsOne)  ia[i]--;  }  fin.close();  fin.open(folderName + "/" + "jg.txt");  ja.resize(ia.back());  for (int i = 0; i < ja.size(); ++i) {  fin >> ja[i];  if (firstNumberIsOne)  ja[i]--;  }  fin.close();  fin.open(folderName + "/" + "ggl.txt");  al.resize(ia.back());  for (int i = 0; i < al.size(); ++i) {  fin >> al[i];  }  fin.close();  fin.open(folderName + "/" + "ggu.txt");  au.resize(ia.back());  for (int i = 0; i < au.size(); ++i) {  fin >> au[i];  }  fin.close();  fin.open(folderName + "/" + "pr.txt");  F.resize(n);  for (int i = 0; i < F.size(); ++i) {  fin >> F[i];  }  fin.close();  return 0;  }  // Ввод СЛАУ: матрицы A и вектора y  void SLAE::writeSLAEToFiles(const string &folderName) {  std::ofstream fout;  fout.open(folderName + "/" + "kuslau.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1)  << n << endl  << maxiter << endl  << E << endl;  fout.close();  fout.open(folderName + "/" + "di.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < di.size(); ++i) {  fout << di[i] << endl;  }  fout.close();  fout.open(folderName + "/" + "ig.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < ia.size(); ++i) {  fout << ia[i] << endl;  }  fout.close();  fout.open(folderName + "/" + "jg.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < ja.size(); ++i) {  fout << ja[i] << endl;  }  fout.close();  fout.open(folderName + "/" + "ggl.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < al.size(); ++i) {  fout << al[i] << endl;  }  fout.close();  fout.open(folderName + "/" + "ggu.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < au.size(); ++i) {  fout << au[i] << endl;  }  fout.close();  fout.open(folderName + "/" + "pr.txt");  fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  for (int i = 0; i < F.size(); ++i) {  fout << F[i] << endl;  }  fout.close();  }  // Вывод плотной матрицы L  void SLAE::writeDenseMatrixLToFile(std::ofstream& fout, const char \*str) {  //fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  fout << str;  for (int i = 0; i < L.size(); ++i) {  for (int j = 0; j < L.size(); ++j) {  fout << L[i][j] << " ";  }  fout << ";" << endl;  }  fout << "]" << endl << endl;  }  // Вывод плотной матрицы U  void SLAE::writeDenseMatrixUToFile(std::ofstream& fout, const char \*str) {  //fout << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  fout << str;  for (int i = 0; i < U.size(); ++i) {  for (int j = 0; j < U.size(); ++j) {  fout << U[i][j] << " ";  }  fout << ";" << endl;  }  fout << "]" << endl << endl;  }  // Преобразуем разряженную матрицу в плотный формат  void SLAE::convAToDense() {  L.clear();  L.resize(n);  for (int i = 0; i < n; ++i) {  L[i].resize(n, 0);  }  for (int i = 0; i < n; ++i) {  L[i][i] = di[i];  int i0 = ia[i];  int i1 = ia[i + 1];  for (int k = i0; k < i1; ++k) { // Идём по всему профилю  L[i][ja[k]] = al[k];  L[ja[k]][i] = au[k];  }  }  }  // Преоразуем матрицы L и U в плотный формат  void SLAE::convLUToDense() {  L.clear();  L.resize(n);  U.clear();  U.resize(n);  for (int i = 0; i < n; ++i) {  L[i].resize(n, 0);  U[i].resize(n, 0);  }  for (int i = 0; i < n; ++i) {  L[i][i] = di\_f[i];  U[i][i] = di\_f[i];  int i0 = ia[i];  int i1 = ia[i + 1];  for (int k = i0; k < i1; ++k) {  L[i][ja[k]] = al\_f[k];  U[ja[k]][i] = au\_f[k];  }  }  }  // A\*x = F  void SLAE::multAAndX() {  F = multA(x);  }  // A\*x = b, где x - произвольный вектор  vec SLAE::multA(const vec&x) {  vec result(n);  for (int i = 0; i < n; ++i) {  result[i] = di[i] \* x[i];  int i0 = ia[i];  int i1 = ia[i + 1];  for (int k = i0; k < i1; ++k) {  result[i] += al[k] \* x[ja[k]];  result[ja[k]] += au[k] \* x[i];  }  }  return result;  }  // A\*x = b, где x - произвольный вектор  vec SLAE::multD(const vec&x) {  vec result(n);  for (int i = 0; i < n; ++i)  result[i] = di\_f[i] \* x[i];  return result;  }  // Создаём вектор x\* = (1,2,...n)'  void SLAE::generateVectX(int size) {  x.resize(size);  for (int i = 0; i < size; ++i) {  x[i] = i + 1;  }  }  // Вывод вектора F в файл  void SLAE::writeFToFile(const char \*fileName) {  std::ofstream fout;  fout.open(fileName);  for (int i = 0; i < F.size(); ++i)  fout << F[i] << endl;  fout.close();  }  // Вывод вектора x в файл  void SLAE::writeXToFile(const char \* fileName) {  std::ofstream fout;  fout.open(fileName);  for (int i = 0; i < x.size(); ++i)  fout << x[i] << " ";  fout << " \t";  fout.close();  }  // Вывод вектора x в поток  void SLAE::writeXToStream(std::ofstream& fout) {  for (int i = 0; i < x.size(); ++i)  fout << x[i] << "\n";  fout << "\n";  }  // Диагональное предобуславливание M = D  void SLAE::decomposionD() {  di\_f.clear();  di\_f.resize(n);  for (int i = 0; i < n; ++i)  di\_f[i] = 1.0 / sqrt(di[i]);  }  // LU\_sq разложение матрицы А  void SLAE::decomposionLUsq() {  real sum\_u, sum\_l, sum\_d;  di\_f = di;  al\_f = al;  au\_f = au;  // Идём построчно в верхнем треугольнике, что экививалентно  // Обходу нижнего треугольника по столбцам вниз, начиная с первого  for (int i = 0; i < n; ++i) {  int i0 = ia[i];  int i1 = ia[i + 1];  // Рассчёт элементов нижнего треугольника  for (int k = i0; k < i1; ++k) {  int j = ja[k]; // текущий j  int j0 = ia[j]; // i0 строки j  int j1 = ia[j + 1]; // i1 строки j  sum\_l = 0;  sum\_u = 0;  int ki = i0; // Индекс l\_ik  int kj = j0; // Индекс u\_kj  while (ki < k && kj < j1) {  if (ja[ki] == ja[kj]) { // l\_ik \* u\_kj  sum\_l += al\_f[ki] \* au\_f[kj];  sum\_u += au\_f[ki] \* al\_f[kj];  ki++;  kj++;  }  else { // Ищем следующие элементы i и j строки, которые можем перемножить  if (ja[ki] > ja[kj]) kj++;  else ki++;  }  }  al\_f[k] = (al\_f[k] - sum\_l) / di\_f[j];  au\_f[k] = (au\_f[k] - sum\_u) / di\_f[j];  }  // Рассчёт диагонального элемента  sum\_d = 0.0;  for (int k = i0; k < i1; ++k)  sum\_d += al\_f[k] \* au\_f[k];  di\_f[i] = sqrt(di\_f[i] - sum\_d);  }  }  // LL' разложение матрицы А  void SLAE::decomposionChol() {  real sum;  di\_f = di;  al\_f = al;  au\_f = au;  // Идём построчно в верхнем треугольнике, что экививалентно  // Обходу нижнего треугольника по столбцам вниз, начиная с первого  for (int i = 0; i < n; ++i) {  int i0 = ia[i];  int i1 = ia[i + 1];  // Рассчёт элементов нижнего треугольника  for (int k = i0; k < i1; ++k) {  int j = ja[k]; // текущий j  int j0 = ia[j]; // i0 строки j  int j1 = ia[j + 1]; // i1 строки j  sum = 0;  int ki = i0; // Индекс l\_ik  int kj = j0; // Индекс u\_kj  while (ki < k && kj < j1) {  if (ja[ki] == ja[kj]) { // l\_ik \* u\_kj  sum += al\_f[ki] \* al\_f[kj];  ki++;  kj++;  }  else { // Ищем следующие элементы i и j строки, которые можем перемножить  if (ja[ki] > ja[kj]) kj++;  else ki++;  }  }  al\_f[k] = (al\_f[k] - sum) / di\_f[j];  }  // Рассчёт диагонального элемента  sum = 0.0;  for (int k = i0; k < i1; ++k)  sum += al\_f[k] \* al\_f[k];  di\_f[i] = sqrt(di\_f[i] - sum);  }  }  // Прямой ход L y = F ==> y = L^-1 F  vec SLAE::execDirectTraversal(const vec &\_F) {  vec y;  y.resize(n, 0);  for (int i = 0; i < n; ++i) {  real sum = 0;  int i0 = ia[i];  int i1 = ia[i + 1];  for (int k = i0; k < i1; ++k)  sum += al\_f[k] \* y[ja[k]];  y[i] = (\_F[i] - sum) / di\_f[i];  }  return y;  }  // Обратный ход U(sq) x = y ==> x = U(sq)^-1 y  vec SLAE::execReverseTraversal(const vec &\_y) {  vec x, y = \_y;  x.resize(n);  for (int i = n - 1; i >= 0; --i) {  x[i] = y[i] / di\_f[i];  int i0 = ia[i];  int i1 = ia[i + 1];  for (int k = i0; k < i1; ++k)  y[ja[k]] -= au\_f[k] \* x[i];  }  return x;  }  // Генерация матрицы Гильберта  void SLAE::createHilbertMatrix(int size) {  clearAll();  n = size;  di.resize(n);  ia.resize(n + 1);  ja.resize(n\*(n - 1) / 2);  al.resize(n\*(n - 1) / 2);  au.resize(n\*(n - 1) / 2);  ia[0] = 0;  ia[1] = 0;  for (int i = 0; i < ia.size() - 1; ++i)  ia[i + 1] = ia[i] + i;  for (int i = 0; i < n; ++i) {  int i0 = ia[i];  int i1 = ia[i + 1];  int j = 0;  for (int k = i0; k < i1; ++k, ++j) {  ja[k] = j;  al[k] = 1.0 / real(i + j + 1);  au[k] = 1.0 / real(i + j + 1);  }  di[i] = 1.0 / real(i + j + 1);  }  }  // Генерация матриц Гильберта  void SLAE::createHilbertMatricies(int a, int b, int step, const string &folderNameTemplate) {  for (int i = a; i <= b; i += step) {  string folderName = folderNameTemplate + to\_string(i);  createHilbertMatrix(i);  generateVectX(i);  multAAndX();  setE(1e-22);  setMaxiter(10000);  writeSLAEToFiles(folderName);  }  }  // Полная очистка СЛАУ  void SLAE::clearAll() {  n = 0;  E = 0.0;  maxiter = 0;  di.clear();  ia.clear();  ja.clear();  al.clear();  au.clear();  di\_f.clear();  al\_f.clear();  au\_f.clear();  x.clear();  r.clear();  z.clear();  p.clear();  F.clear();  Ftmp.clear();  }  // Вычисление нормы в Евклидовом пространстве  real SLAE::calcNormE(vec &x) {  return sqrt(x \* x);  }  // Рассчёт относительной невязки  real SLAE::calcRelativeDiscrepancy() {  //return calcNormE(r) / calcNormE(F);  return (r\*r);  }  // Локально - оптимальная схема  pair<int, real> SLAE::LOS() {  x.clear(); // Задаём начальное приближение  x.resize(n, 0); // x\_0 = (0, 0, ...)  r.resize(n);  vec xprev = x;  r = F - multA(x); // r\_0 = f - A\*x\_0  z = r; // z\_0 = r\_0  p = multA(z); // p\_0 = A\*z\_0  for (int i = 0; i < maxiter; ++i) {  real pp = (p \* p);  real alpha = (p \* r) / pp;  x = x + alpha \* z;  r = r - alpha \* p;  Ftmp = multA(r);  real beta = -(p \* Ftmp) / pp;  z = r + beta \* z;  p = Ftmp + beta \* p;  real relativeDiscrepancy = calcRelativeDiscrepancy();  if (x == xprev || relativeDiscrepancy < E)  return make\_pair(i, relativeDiscrepancy);  xprev = x;  }  }  // Локально - оптимальная схема c неполной диагональной факторизацией  pair<int, real> SLAE::LOSfactD() {  x.clear(); // Задаём начальное приближение  x.resize(n, 0); // x\_0 = (0, 0, ...)  vec xprev = x;  decomposionD();  r = F - multA(x); // r\_0 = f - A\*x\_0  r = multD(r);  z = multD(r); // z = U^-1 r  p = multA(z); // p = A\*z  p = multD(p); // p = L^-1 A\*z  for (int i = 0; i < maxiter; ++i) {  real pp = p \* p;  real alpha = (p\*r) / pp;  x = x + alpha \* z;  r = r - alpha \* p;  vec tmp = multD(r);  tmp = multA(tmp);  tmp = multD(tmp);  real beta = -(p \* tmp) / pp;  p = tmp + beta \* p;  tmp = multD(r);  z = tmp + beta \* z;  real relativeDiscrepancy = calcRelativeDiscrepancy();  if (x == xprev || relativeDiscrepancy < E)  return make\_pair(i, relativeDiscrepancy);  xprev = x;  }  }  // Локально - оптимальная схема с неполной факторизацией LU(sq)  pair<int, real> SLAE::LOSfactLUsq() {  x.clear(); // Задаём начальное приближение  x.resize(n, 0); // x\_0 = (0, 0, ...)  vec xprev = x;  decomposionLUsq();  r = F - multA(x); // r\_0 = f - A\*x\_0  r = execDirectTraversal(r); // r = L^-1 (f - A\*x\_0)  z = execReverseTraversal(r); // z = U^-1 r  p = multA(z); // p = A\*z  p = execDirectTraversal(p); // p = L^-1 A\*z  for (int i = 0; i < maxiter; ++i) {  real pp = p \* p;  real alpha = (p\*r) / pp;  x = x + alpha \* z;  r = r - alpha \* p;  vec tmp = execReverseTraversal(r);  tmp = multA(tmp);  tmp = execDirectTraversal(tmp);  real beta = -(p \* tmp) / pp;  p = tmp + beta \* p;  tmp = execReverseTraversal(r);  z = tmp + beta \* z;  real relativeDiscrepancy = calcRelativeDiscrepancy();  if (x == xprev || relativeDiscrepancy < E)  return make\_pair(i, relativeDiscrepancy);  xprev = x;  }  } |

**main.cpp**

|  |
| --- |
| #include "slae.h"  #include <iomanip>  void testSLAE(const string &folderName, bool firstNumberIsOne, bool doWriteHeader) {  SLAE slae;  pair <int, real> iterationsCountAndDiscrapancy;  std::ofstream foutTable, foutX;  foutTable.open(folderName + "/table.txt");  foutX.open(folderName + "/x.txt");  foutTable << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1) << std::scientific;  foutX << std::fixed << std::setprecision(std::numeric\_limits<real>::digits10 + 1);  if (doWriteHeader)  foutTable << "Method\tIterations count\tTime\tRelative discrepancy" << endl;  slae.clearAll();  slae.readSLAEfromFiles(folderName, firstNumberIsOne);  auto begin = std::chrono::steady\_clock::now();  iterationsCountAndDiscrapancy = slae.LOS();  auto end = std::chrono::steady\_clock::now();  auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);  foutTable << "LOS\t" << iterationsCountAndDiscrapancy.first << "\t" << elapsed\_ms.count() << "\t" << iterationsCountAndDiscrapancy.second << endl;  slae.writeXToStream(foutX);  slae.clearAll();  slae.readSLAEfromFiles(folderName, firstNumberIsOne);  begin = std::chrono::steady\_clock::now();  iterationsCountAndDiscrapancy = slae.LOSfactD();  end = std::chrono::steady\_clock::now();  elapsed\_ms = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);  foutTable << "LOS + diag\t" << iterationsCountAndDiscrapancy.first << "\t" << elapsed\_ms.count() << "\t" << iterationsCountAndDiscrapancy.second << endl;  slae.writeXToStream(foutX);  slae.clearAll();  slae.readSLAEfromFiles(folderName, firstNumberIsOne);  begin = std::chrono::steady\_clock::now();  iterationsCountAndDiscrapancy = slae.LOSfactLUsq();  end = std::chrono::steady\_clock::now();  elapsed\_ms = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);  foutTable << "LOS + LU(sq)\t" << iterationsCountAndDiscrapancy.first << "\t" << elapsed\_ms.count() << "\t" << iterationsCountAndDiscrapancy.second << endl;  slae.writeXToStream(foutX);  foutTable.close();  foutX.close();  }  int main() {  // Сначала нужно создать папки HilbertN, N - размерность матрицы  /\*SLAE slae;  slae.createHilbertMatricies(4, 12, 4, "Hilbert");\*/  testSLAE("A", false, false);  testSLAE("B", false, false);  testSLAE("Hilbert4", false, false);  testSLAE("Hilbert8", false, false);  testSLAE("Hilbert12", false, false);  testSLAE("0945", true, false);  testSLAE("4545", true, false);  } |